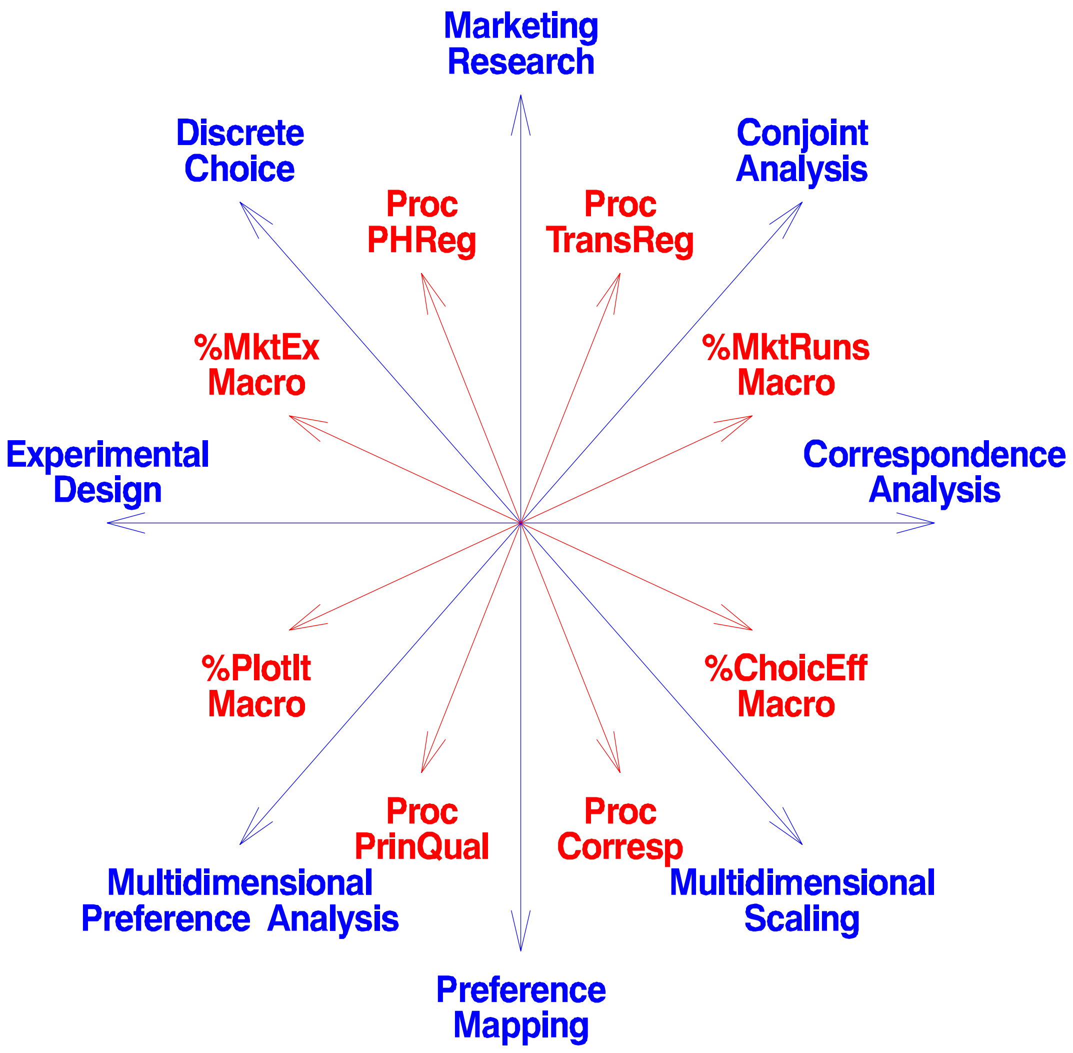
Marketing Research Methods in SAS

Experimental Design, Choice, Conjoint, and Graphical Techniques



Warren F. Kuhfeld

October 1, 2010
SAS 9.2 Edition
MR-2010

Contents Overview

Contents

Preface

Marketing Research Methods in SAS discusses experimental design, discrete choice, conjoint analysis, and graphical and perceptual mapping techniques. The book has grown and evolved over many years and many revisions. For example, the section on choice models grew from a two-page handout written by Dave DeLong in 1992. This edition was written for SAS 9.2 and subsequent SAS releases.

This book was written for SAS macros that are virtually identical to those shipped with the SAS 9.22 release in 2010. All of the macros and most of the code used in this book should work in SAS 9.0, 9.1, and SAS 9.2. However, some features, such as the standardized orthogonal contrast coding in the `%ChoicEff` macro, require SAS 9.2 or a later release. To be absolutely sure that you have the macros that correspond to this book, you should get the latest macros from the Web. All other macros are obsolete. Copies of this book and all of the macros are available on the Web (reports beginning with "MR-2010" at `http://support.sas.com/resources/papers/tnote/tnote_marketresearch.html`). This book is the October 1, 2010 edition, and it uses the macros that are dated July 25, 2010.

I hope that this book and tool set will help you do better research, do it quickly, and do it more easily. I would like to hear what you think. Many of my examples and enhancements to the software are based on feedback from people like you. If you would like to be added to a mailing list to receive periodic e-mail updates on SAS marketing research tools (probably no more than once every few months), e-mail Warren.Kuhfeld at sas.com. This list will not be sold or used for any other purpose.

Finishing a 1309-page book causes one to pause and reflect. As always, I am proud of this edition of the book and tools, however it is clear that I have stood on the shoulders of giants. The following people contributed to writing portions of this book: Mark Garratt, Joel Huber, Ying So, Randy Tobias, Wayne Watson, and Klaus Zwerina. My parts could not have been written without the help of many people. I would like to thank Joel Huber, Ying So, Randy Tobias, and John Wurst. My involvement in the area of experimental design and choice modeling can be traced to several conversations with Mark Garratt in the early 1990's and then to the influence of Don Anderson, Joel Huber, Jordan Louviere, and Randy Tobias. I first learned about choice modeling at a tutorial taught by Jordan Louviere at the ART Forum. Later, as I got into this area, Jordan was very helpful at key times in my professional development. Don Anderson has been a great friend and influence over the years. Don did so much of the pioneering work on choice designs. There is no doubt that his name should be referenced in this book way more than it is. Joel Huber got me started on the work that became the `%ChoicEff` macro. Randy Tobias has been a great colleague and a huge help to me over the years in all areas of experimental design, and many components of the `%MktEx` macro and other design macros are based on his ideas and his work. Randy wrote PROC OPTEX and PROC FACTEX which provide the foundation for my design work. My work on balanced incomplete block designs can be traced to conversations with John Wurst.

Don Anderson, Warwick de Launey, Nam-Ky Nguyen, Shanqi Pang, Neil Sloane, Chung-yi Suen, Randy Tobias, J.C. Wang, and Yingshan Zhang kindly helped me with some of the orthogonal arrays in the `%MktEx` macro. Brad Jones advised me on coordinate exchange. Much of our current success with creating highly restricted designs is due to the difficult and very interesting design problems brought to me by Johnny Kwan. I have also learned a great deal from the interesting and challenging problems brought to me by Ziad Elmously.

There are a few other people that I would like to acknowledge. Without these people, I would have never been in the position to write a book such as this. From my undergraduate days at Kent State, I would like to thank Roy Lilly[*], Larry Melamed, Steve Simnick and especially my adviser Ben Newberry. From graduate school at UNC, I would like to thank Ron Helms, Keith Muller, and especially my adviser Forrest Young[*]. From SAS, I would like to thank Bob Rodriguez, Warren Sarle, and all of my colleagues in SAS/STAT Research and Development. It is great to work with such a smart, talented, productive, and helpful group of people.

On a more personal note, I was diagnosed with prostate cancer in 2008. Most prostate cancers are not very aggressive. Someone forgot to tell mine that. My Gleason Score was 9. A Gleason Score is a measure of prostate cancer aggressiveness that ranges from 2 to 10. A 9 is almost as scary as they come. Thanks to modern medicine, early detection, and a brilliant and gifted surgeon using the latest technology, I am doing very well. Advocates of early testing and screening are trying to catch cases like mine early, while there is still time for a cure. In my case, every indication is that they were successful and surgery alone got it all. I get my PSA checked every three months now, and PSA since the surgery has consistently been undetectable, which is perfect. I have been cancer free for over two years now and am in the best shape of my life. I hope that all of you, men and women, get your regular physical exams and health screenings and see your health care provider if you notice any changes in your body and how it functions. Yes, I know it's not fun. Do it anyways! It saved my life; it might save yours too. I would like to thank a few of my friends who helped me through this period and the other difficult times that I went through in that year: Woody, Mike, Sara, Benny, Deborah, Gina, and Peg. You are my guardian angels. You gave me hope, help, and support, and you were there when I needed you the most.

Finally, I would like to thank my mother[*], my father[*], my sister, and my stepfather Ed[*], for being so good to my Mom and for being such a wonderful grandfather to my children. I dedicate this edition of the book to my children, Megan and Rusty, and to Donna, who helped me learn how to live and love again.

Warren F. Kuhfeld, Ph.D.
Manager, Multivariate Models R&D
SAS Institute Inc.
October 1, 2010

[*]It is sad that so many people that I acknowledge have passed away since I started working on this book. I wish I could thank all of these people for their role in helping me to get to where I am today.

About this Edition

The 2010 edition of **Marketing Research Methods in SAS** is a partial revision of the 2009 book. I did not have time to rewrite everything that I would have liked to rewrite. I do many different things professionally, way more than most readers of this book know. Those other things take most of my time, and it is hard to find the large block of time that I need to completely modify a piece of work this size every time there is an enhancement or innovation in the design macros. In this edition, I added new material and also added some guidance in the ensuing paragraphs about how to navigate through this book.

This edition has explicit instructions about how to contact Technical Support when you have questions or problems. See page 25 for more information. While I have never minded getting your questions, they really need to go to Technical Support first. I am not always in the office. Sometimes I am out backpacking without any contact with the outside world. Contacting Technical Support will ensure that your question is seen and addressed in a timely manner.

This edition contains some major new features that were not in the 2005 edition and one major new feature that was not in the 2009 edition. With this 2010 edition, the `%ChoicEff` macro now allows you to specify a restrictions macro. You can use it to specify within alternative restrictions, within choice set (and across alternative) restrictions, and even restrictions across choice sets. You can specify restrictions directly with the alternative-swapping algorithm. You no longer need to make a choice design with the `%MktEx` macro or with the choice-set-swapping algorithm in the `%ChoicEff` macro when there are restrictions.

Most of this book is about experimental design. In particular, most of it is about designing choice experiments. This is a big topic with multiple tools and multiple approaches with multiple nuances, so hundreds of pages are devoted to it. This can be intimidating when you are first getting started. The following information can help you get started:

- If you are new to choice modeling and choice design, and you want to understand what you are doing, you should start by reading the "Experimental Design: Efficiency, Coding, and Choice Designs" chapter, which starts on page 53. It is a self-contained short course on basic choice design, complete with exercises at the end.
- If you just want to jump in and get started designing experiments, see the examples of the `%ChoicEff` macro starting on page 808. This section describes all of the tools that you need to design almost any choice experiment. Many other tools and approaches exist and are described in detail elsewhere in the book, but you almost certainly can get by with the subset described starting on page 808. However, if you are going to approach choice modeling intelligently, you need to understand the coding and modeling issues discussed in the experimental design chapter and elsewhere throughout this book.
- If you want to understand the choice model and the classic approach to choice design, see the "Discrete Choice" chapter starting on page 285. While this chapter contains lots of great information on many topics related to choice modeling, and it uses an approach in most examples that is in many cases optimal or at least good, most of that chapter uses an approach that seems to be less often used now days.

The process of designing an experiment for a linear model is generally straight-forward since software, such as the `%MktEx` macro, exists for finding an optimal (or at least efficient) design for the specified model. In contrast, the process of designing a choice experiment is guided more by heuristics than hard science. You can only design an optimal experiment for a choice model if you know the parameters, and if you knew the parameters, there would be no reason to design the experiment. Much of the early work in choice design took a linear model design approach, which is discussed in detail in the design chapter starting on page 53 and the "Discrete Choice" chapter starting on page 285. In this approach, you make a design that is orthogonal and balanced (or at least nearly so) in all of the attributes of all of the alternatives and rearrange that into a choice design. This approach has much to recommend it, particularly in the context of alternative-specific designs and designs with complicated effects such as availability and cross effects. It is not the optimal approach for generic designs and simpler design problems.

In previous editions, I referred to this approach to designing choice experiments as the "linear design" approach. With this edition, I have banished that phrase from this book. That phrase has always been problematic and confusing. With this edition, I now use phrases like "linear model design" and "factorial design" interchangeably to refer to designs that will be used for a linear model such as a conjoint analysis. I no longer refer to a design constructed by the `%MktEx` macro that is converted to a choice design by the `%MktRoll` macro as a "linear design." Instead, I use the term "linear arrangement" as a short-hand for "linear arrangement of a choice design" to refer to a design that will ultimately be used for a choice design, but is currently arranged with one row per choice set and one column for every attribute of every alternative. The linear arrangement of a choice design can be constructed and evaluated by pretending that it will be used for a linear model with one factor for every attribute of every alternative. This is one way in which you can make a choice design, and it is discussed in detail in this book.

If you had to pick one approach to solve all of your design problems, and you did not have time to learn about all of the other ways you could go about designing a choice experiment, here is what I would recommend. Use the `%MktEx` macro to make a candidate set of alternatives, and use the `%ChoicEff` macro to create a choice design from it. If there are any restrictions on your design, use the `restrictions=` option in the `%ChoicEff` macro to impose the restrictions. The `restrictions=` option in the `%ChoicEff` macro is new with this edition of the book and macros. Restrictions can be within alternative, within choice set (and across alternative), or even across choice sets. You can impose restrictions to prevent certain combinations of alternatives from occurring together, to minimize the burden on the subjects, to eliminate dominated alternatives, to make the design more realistic, or for any other reason. I have not eliminated the hundreds of pages of this book that are devoted to other ways to make choice designs, because those pages contain a lot of useful information. Rather, I simply point out that you can selectively devote your attention to different parts of the book and concentrate on using the `%ChoicEff` macro with a candidate set of alternatives for most of your choice design needs.

Each of the last few editions has relied much more heavily on the `%ChoicEff` macro than preceding editions did. The `%ChoicEff` macro is heavily used both for design construction and for design evaluation. You should always use it to evaluate designs before data are collected. This has always been good advice, but with the addition of the standardized orthogonal contrast coding in PROC TRANSREG (which the macro calls) plus some new options and output, the `%ChoicEff` macro now provides a clearer picture of choice design goodness for many choice designs. In particular, it provides a measure of design efficiency on a 0 to 100 scale for at least some choice designs. See page 81 for more information.

A big part of this book is about experimental design. Efficient experimental-design software, like some other search software, is notorious for not finding the exact same results if anything changes (operating system, computer, SAS release, code version, compiler, math library, phase of the moon, and so on), and the `%MktEx` and `%ChoicEff` macros are no exception. They will find the same design if you specify a random number seed and run the same macro over and over again on the same machine, but if you change anything, they might find a different design. The algorithms are seeking to optimize an efficiency function. All it takes is one little difference, such as two numbers being almost identical but different in the last bit, and the algorithm can start down a different path. We expect as things change and the code is enhanced that the designs will be similar. Sometimes two designs might even have the exact same efficiency, but they will not be identical. The `%MktEx` and `%ChoicEff` macros, and other efficient design software take every step that increases efficiency. One can envision an alternative algorithm that repeatedly evaluates every possible step and then takes only the largest one with fuzzing to ensure proper tie handling. Such an algorithm would be less likely to give different designs, but it would be *much* slower. Hence, we take the standard approach of using a fast algorithm that makes great designs, but not always the same designs.

For many editions, I regenerated every design, every sample data set, every bit of output, and then made changes all over the text to refer to the new output. Many times I had to do this more than once when a particularly attractive enhancement that changed the results occurred to me late in the writing cycle. It was difficult, tedious, annoying, error prone, and time consuming, and it really did not contribute much to the book since you would very likely be running under a different configuration than me and not get exactly the same answers as me, no matter what either you or I did. Starting with the January 2004 edition, I said enough is enough! For many versions now, in the accompanying sample code, I have hard-coded in the actual example design after the code so you can run the sample and reproduce my results. I am continuing to do that, however I have not redone every example. Expect to get similar but different results, and use the sample code if you want to get the exact same design that was in the book. I would rather spend my time giving you new capabilities than rewriting old examples that have not changed in any important way.

In this and every other edition, all of the data sets in the discrete choice and conjoint examples are artificial. As a software developer, I do not have access to real data. Even if I did, it would be hard to use them since most of those chapters are about design. Of course the data need to come from subjects who make judgments based on the actual design. If I had real data in an example, I would no longer be able to change and enhance the design strategy for that example. Many of the examples have changed many times over the years as better design software and strategies became available. In this edition, like all previous editions, the emphasis is on showing design strategies not on illustrating the analysis of the data.

The orthogonal array catalog is essentially complete up through 143 runs,[*] with pretty good coverage from 144 to 513 runs, and spotty coverage beyond 513 runs. New arrays are being discovered regularly. If you know of any orthogonal arrays that are not in my catalog, please e-mail Warren.Kuhfeld at sas.com. I would particularly like to hear from you if you know how to make any of the arrays that are missing. Also, if you know how to construct any of these difference schemes, I would appreciate hearing from you: D(60, 36, 3); D(102, 51, 3); D(60, 21, 4); D(112, 64, 4); D(30, 15, 5); D(35, 17, 5); D(40, 25, 5); D(55, 17, 5); D(60, 25, 5); D(65, 25, 5); D(85, 35, 5); D(60, 11, 6); D(84, 16, 6); D(35, 11, 7); D(63, 28, 7); D(40, 8, 10); and D(30, 7, 15). The notation D(r, c, s) refers to an $r \times c$ matrix of order s. You can always go to `http://support.sas.com/techsup/technote/ts723.html` to see the current state of the orthogonal array catalog.

[*]There are a few missing designs in 108 runs. I would welcome help in making them.

ODS Graphics is used throughout the book. With ODS Graphics and SAS 9.2, statistical procedures produce graphs as automatically as they produce tables, and graphs are now integrated with tables in the ODS output. See 1247 for the section of the book that says the most about ODS Graphics. Also see “Chapter 21, Statistical Graphics Using ODS” in SAS/STAT documentation for more on ODS Graphics: `http://support.sas.com/documentation/`. You can learn more about ODS Graphics in my new book, **Statistical Graphics in SAS: An Introduction to the Graph Template Language and the Statistical Graphics Procedures**. You can learn more about the book at `http://support.sas.com/publishing/authors/kuhfeld.html`.

I hope you like this edition. Feedback is welcome. Your feedback can help make these tools better.

Getting Help and Contacting Technical Support

SAS Technical Support can help you if you encounter a problem or issue while working with the market research design macros or procedures in this book. However, you can help Technical Support greatly by providing certain details of your problem.

A new track will be initiated when you contact Technical Support about a specific problem, and notes added to that track as you work through the problem with your support specialist. For this reason, you should avoid starting multiple tracks on the same topic.

You can expect to hear back from a support specialist within one business day, but this does not necessarily mean that your question will be resolved by then. You might be asked to provide additional information to help solve your problem.

Opening a Track via the Web

You can contact Technical Support at the Technical Support Web site, which can be opened by using the link below. Working through a problem with your technical support specialist via Web and email is recommended for usage questions relating to this book.

`http://support.sas.com/ctx/supportform/index.jsp`

Opening a Track via the Phone

You can contact SAS Technical support via phone. We recommend this approach for short questions only. Please consult the SAS Technical Support Web site by clicking on the link below to obtain the appropriate Technical Support phone numbers for US and international users.

SAS Support Phone Numbers
919.677.8008 (US)
`http://www.sas.com/offices/intro.html` (International Support via Worldwide SAS Offices)

Important Information to Provide SAS Technical Support

Providing the following pieces of information to Technical Support can significantly shorten the time necessary to understand and solve your problem:

• **Your Contact Information.** Provide your full contact information: name, phone number, email address, and site number.

• **Information about your SAS Version and Market Design Macros.** Please include information about the version of SAS that you have installed and are using. You can find this information under Help → About SAS.

Please include information about the version of the macros that you have installed and are using. You can find this information by submitting the following statement before running any of the macros: `%let mktopts = version;`.

Example:

```
    1? %let mktopts = version;
    2? %mktex(2 ** 3, n=4)
```

Produces:

```
  MktEx macro version 25Jul2010
  MktRuns macro version 25Jul2010
  Seed = 4247959
  MktOrth macro version 25Jul2010
```

Note that some macros call other macros, and all must be the same version.

• **Information about your Design.** Please describe your design fully:

1. identify the type of design you want to generate (for example, choice, MaxDiff, conjoint, partial profile)
2. the number of factors, the number of levels associated with each factors
3. the number of runs (or choice sets) in the final design
4. the number of alternatives in a choice design
5. the model you want to estimate
6. if your model has constraints, define the desired constraints

• **Details about your Problem.** Include the program statements that you have tried to generate the design. Did you see an warning or error message in connection with your problem? If so, please attach a copy of the message to your technical support inquiry, and include a copy of the SAS .log file for the analysis.

Marketing Research: Uncovering Competitive Advantages

Warren F. Kuhfeld

Abstract

SAS provides a variety of methods for analyzing marketing data including conjoint analysis, correspondence analysis, preference mapping, multidimensional preference analysis, and multidimensional scaling. These methods allow you to analyze purchasing decision trade-offs, display product positioning, and examine differences in customer preferences. They can help you gain insight into your products, your customers, and your competition. This chapter discusses these methods and their implementation in SAS.[*]

Introduction

Marketing research is an area of applied data analysis whose purpose is to support marketing decision making. Marketing researchers ask many questions, including:

- Who are my customers?
- Who else should be my customers?
- Who are my competitors' customers?
- Where is my product positioned relative to my competitors' products?
- Why is my product positioned there?
- How can I reposition my existing products?
- What new products should I create?
- What audience should I target for my new products?

[*]Copies of this chapter (MR-2010A), the other chapters, sample code, and all of the macros are available on the Web `http://support.sas.com/resources/papers/tnote/tnote_marketresearch.html`. This is a minor modification of a paper that was presented to SUGI 17 by Warren F. Kuhfeld and to the 1992 Midwest SAS Users Group meeting by Russell D. Wolfinger.

Marketing researchers try to answer these questions using both standard data analysis methods, such as descriptive statistics and crosstabulations, and more specialized marketing research methods. This chapter discusses two families of specialized marketing research methods, perceptual mapping and conjoint analysis. Perceptual mapping methods produce plots that display product positioning, product preferences, and differences between customers in their product preferences. Conjoint analysis is used to investigate how consumers trade off product attributes when making a purchasing decision.

Perceptual Mapping

Perceptual mapping methods, including correspondence analysis (CA), multiple correspondence analysis (MCA), preference mapping (PREFMAP), multidimensional preference analysis (MDPREF), and multidimensional scaling (MDS), are data analysis methods that generate graphical displays from data. These methods are used to investigate relationships among products as well as individual differences in preferences for those products.[*]

CA and MCA can be used to display demographic and survey data. CA simultaneously displays in a scatter plot the row and column labels from a two-way contingency table (crosstabulation) constructed from two categorical variables. MCA simultaneously displays in a scatter plot the category labels from more than two categorical variables.

MDPREF displays products positioned by overall preference patterns. MDPREF also displays differences in how customers prefer products. MDPREF displays in a scatter plot both the row labels (products) and column labels (consumers) from a data matrix of continuous variables.

MDS is used to investigate product positioning. MDS displays a set of object labels (products) whose perceived similarity or dissimilarity has been measured.

PREFMAP is used to interpret preference patterns and help determine why products are positioned where they are. PREFMAP displays rating scale data in the same plot as an MDS or MDPREF plot. PREFMAP shows both products and product attributes in one plot.

MDPREF, PREFMAP, CA, and MCA are all similar in spirit to the biplot, so first the biplot is discussed to provide a foundation for discussing these methods.

The Biplot. A *biplot* (Gabriel 1981) simultaneously displays the row and column labels of a data matrix in a low-dimensional (typically two-dimensional) plot. The "bi" in "biplot" refers to the *joint* display of rows and columns, not to the dimensionality of the plot. Typically, the row coordinates are plotted as points, and the column coordinates are plotted as vectors.

Consider the artificial preference data matrix in Figure 1. Consumers were asked to rate their preference for products on a 0 to 9 scale where 0 means little preference and 9 means high preference. Consumer 1's preference for Product 1 is 4. Consumer 1's most preferred product is Product 4, which has a preference of 6.

[*]Also see pages 1231 and 1263.

$$\begin{array}{l|ccc|} & \text{Consumer 1} & \text{Consumer 2} & \text{Consumer 3} \\ \text{Product 1} & 4 & 1 & 6 \\ \text{Product 2} & 4 & 2 & 4 \\ \text{Product 3} & 1 & 0 & 2 \\ \text{Product 4} & 6 & 2 & 8 \end{array}$$

Figure 1. Preference Data Matrix

$$\begin{array}{ccccc} Y & = & A & \times & B' \\ \begin{bmatrix} 4 & 1 & 6 \\ 4 & 2 & 4 \\ 1 & 0 & 2 \\ 6 & 2 & 8 \end{bmatrix} & = & \begin{bmatrix} 1 & 2 \\ 2 & 0 \\ 0 & 1 \\ 2 & 2 \end{bmatrix} & \times & \begin{bmatrix} 2 & 1 & 2 \\ 1 & 0 & 2 \end{bmatrix} \end{array}$$

Figure 2. Preference Data Decomposition

The biplot is based on the idea of a matrix decomposition. The $(n \times m)$ data matrix $\mathbf{Y}$ is decomposed into the product of an $(n \times q)$ matrix $\mathbf{A}$ and a $(q \times m)$ matrix $\mathbf{B}'$. Figure 2 shows a decomposition of the data in Figure 1.[*] The rows of $\mathbf{A}$ are coordinates in a two-dimensional plot for the row points in $\mathbf{Y}$, and the columns of $\mathbf{B}'$ are coordinates in the same two-dimensional plot for the column points in $\mathbf{Y}$. In this artificial example, the entries in $\mathbf{Y}$ are exactly reproduced by *scalar products* of coordinates. For example, the $(1, 1)$ entry in $\mathbf{Y}$ is $y_{11} = a_{11} \times b_{11} + a_{12} \times b_{12} = 4 = 1 \times 2 + 2 \times 1$.

The rank of $\mathbf{Y}$ is $q \leq MIN(n, m)$. The rank of a matrix is the minimum number of dimensions that are required to represent the data without loss of information. The rank of $\mathbf{Y}$ is the full number of columns in $\mathbf{A}$ and $\mathbf{B}$. In the example, $q = 2$. When the rows of $\mathbf{A}$ and $\mathbf{B}$ are plotted in a two-dimensional scatter plot, the scalar product of the coordinates of $\mathbf{a}'_i$ and $\mathbf{b}'_j$ *exactly* equals the data value y_{ij}. This kind of scatter plot is a biplot. When $q > 2$ and the first two dimensions are plotted, then $\mathbf{AB}'$ is *approximately* equal to $\mathbf{Y}$, and the display is an *approximate biplot*.[†] The best values for $\mathbf{A}$ and $\mathbf{B}$, in terms of minimum squared error in approximating $\mathbf{Y}$, are found using a singular value decomposition (SVD).[‡] An approximate biplot is constructed by plotting the first two columns of $\mathbf{A}$ and $\mathbf{B}$.

When $q > 2$, the full geometry of the data cannot be represented in two dimensions. The first two columns of $\mathbf{A}$ and $\mathbf{B}$ provide the best approximation of the high dimensional data in two dimensions. Consider a cloud of data in the shape of an American football. The data are three dimensional. The best one dimensional representation of the data—*the first principal component*—is the line that runs from one end of the football, through the center of gravity or *centroid* and to the other end. It is the longest line that can run through the football. The second principal component also runs through the centroid and is perpendicular or *orthogonal* to the first line. It is the longest line that can be drawn through the centroid that is perpendicular to the first. If the football is a little thicker at the laces, the second principal component runs from the laces through the centroid and to the other side of the football. All of the points in the football shaped cloud can be projected into the plane of the first two principal components. The resulting scatter plot will show the approximate shape of the data. The two longest dimensions are shown, but the information in the other dimensions are lost. This is the principle behind approximate biplots. See Gabriel (1981) for more information about the biplot.

[*]Figure 2 does not contain the decomposition that would be used for an actual biplot. Small integers were chosen to simplify the arithmetic.

[†]In practice, the term biplot is sometimes used without qualification to refer to an approximate biplot.

[‡]SVD is sometimes referred to in the psychometric literature as an Eckart-Young (1936) decomposition. SVD is closely tied to the statistical method of principal component analysis.

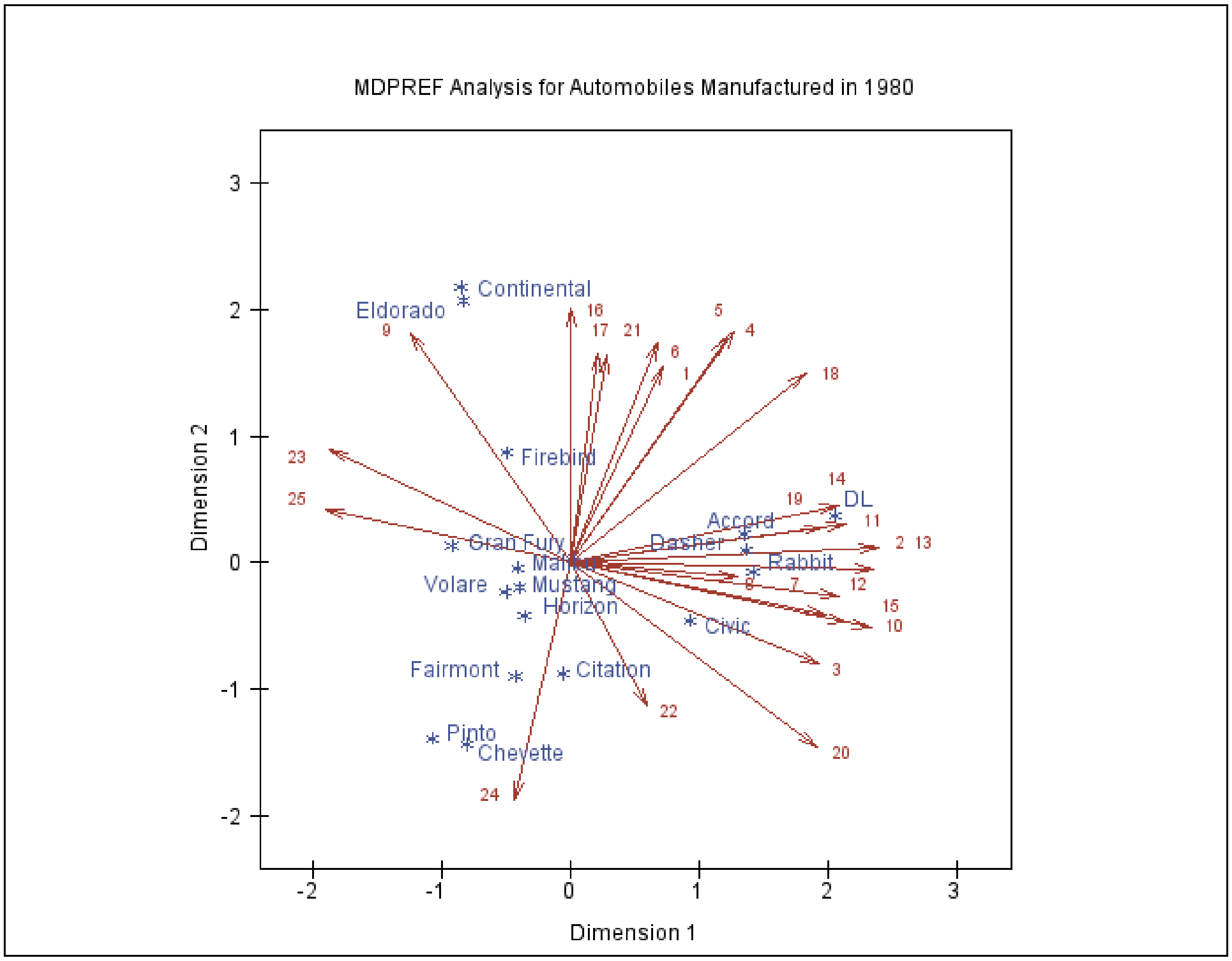


Figure 3. Multidimensional Preference Analysis

Multidimensional Preference Analysis. Multidimensional Preference Analysis (Carroll 1972) or MDPREF is a biplot analysis for preference data. Data are collected by asking respondents to rate their preference for a set of objects—products in marketing research.

Questions that can be addressed with MDPREF analyses include: Who are my customers? Who else should be my customers? Who are my competitors' customers? Where is my product positioned relative to my competitors' products? What new products should I create? What audience should I target for my new products?

For example, consumers were asked to rate their preference for a group of automobiles on a 0 to 9 scale, where 0 means no preference and 9 means high preference. **Y** is an $(n \times m)$ matrix that contains ratings of the n products by the m consumers. Figure 3 displays an example in which 25 consumers rated their preference for 17 new (at the time) 1980 automobiles. Each consumer is a vector in the space, and each car is a point identified by an asterisk (*). Each consumer's vector points in *approximately* the direction of the cars that the consumer most preferred.

The dimensions of this plot are the first two principal components. The plot differs from a proper biplot of **Y** due to scaling factors. At one end of the plot of the first principal component are the most preferred automobiles; the least preferred automobiles are at the other end. The American cars on the average were least preferred, and the European and Japanese cars were most preferred. The second principal component is the longest dimension that is orthogonal to the first principal component. In the example, the larger cars tend to be at the top and the smaller cars tend to be at the bottom.

The automobile that projects farthest along a consumer vector is that consumer's most preferred automobile. To project a point onto a vector, draw an imaginary line through a point crossing the vector at a right angle. The point where the line crosses the vector is the *projection*. The length of this projection differs from the predicted preference, the scalar product, by a factor of the length of the consumer vector, which is constant within each consumer. Since the goal is to look at projections of points onto the vectors, the absolute length of a consumer's vector is unimportant. The relative lengths of the vectors indicate fit, with longer vectors indicating better fit. The coordinates for the endpoints of the vectors were multiplied by 2.5 to extend the vectors and create a better graphical display. The direction of the preference scale is important. The vectors point in the direction of increasing values of the data values. If the data had been ranks, with 1 the most preferred and n the least preferred, then the vectors would point in the direction of the least preferred automobiles.

Consumers 9 and 16, in the top left portion of the plot, most prefer the large American cars. Other consumers, with vectors pointing up and nearly vertical, also show this pattern of preference. There is a large cluster of consumers, from 14 through 20, who prefer the Japanese and European cars. A few consumers, most notably consumer 24, prefer the small and inexpensive American cars. There are no consumer vectors pointing through the bottom left portion of the plot between consumers 24 and 25, which suggests that the smaller American cars are generally not preferred by any of these consumers.

Some cars have a similar pattern of preference, most notably Continental and Eldorado. This indicates that marketers of Continental or Eldorado may want to try to distinguish their car from the competition. Dasher, Accord, and Rabbit were rated similarly, as were Malibu, Mustang, Volare, and Horizon. Several vectors point into the open area between Continental/Eldorado and the European and Japanese cars. The vectors point away from the small American cars, so these consumers do not prefer the small American cars. What car would these consumers like? Perhaps they would like a Mercedes or BMW.

Preference Mapping. Preference mapping[*] (Carroll 1972) or PREFMAP plots resemble biplots, but are based on a different model. The goal in PREFMAP is to project external information into a configuration of points, such as the set of coordinates for the cars in the MDPREF example in Figure 3. The external information can aid interpretation.

Questions that can be addressed with PREFMAP analyses include: Where is my product positioned relative to my competitors' products? Why is my product positioned there? How can I reposition my existing products? What new products should I create?

[*]Preference mapping is sometimes referred to as external unfolding.

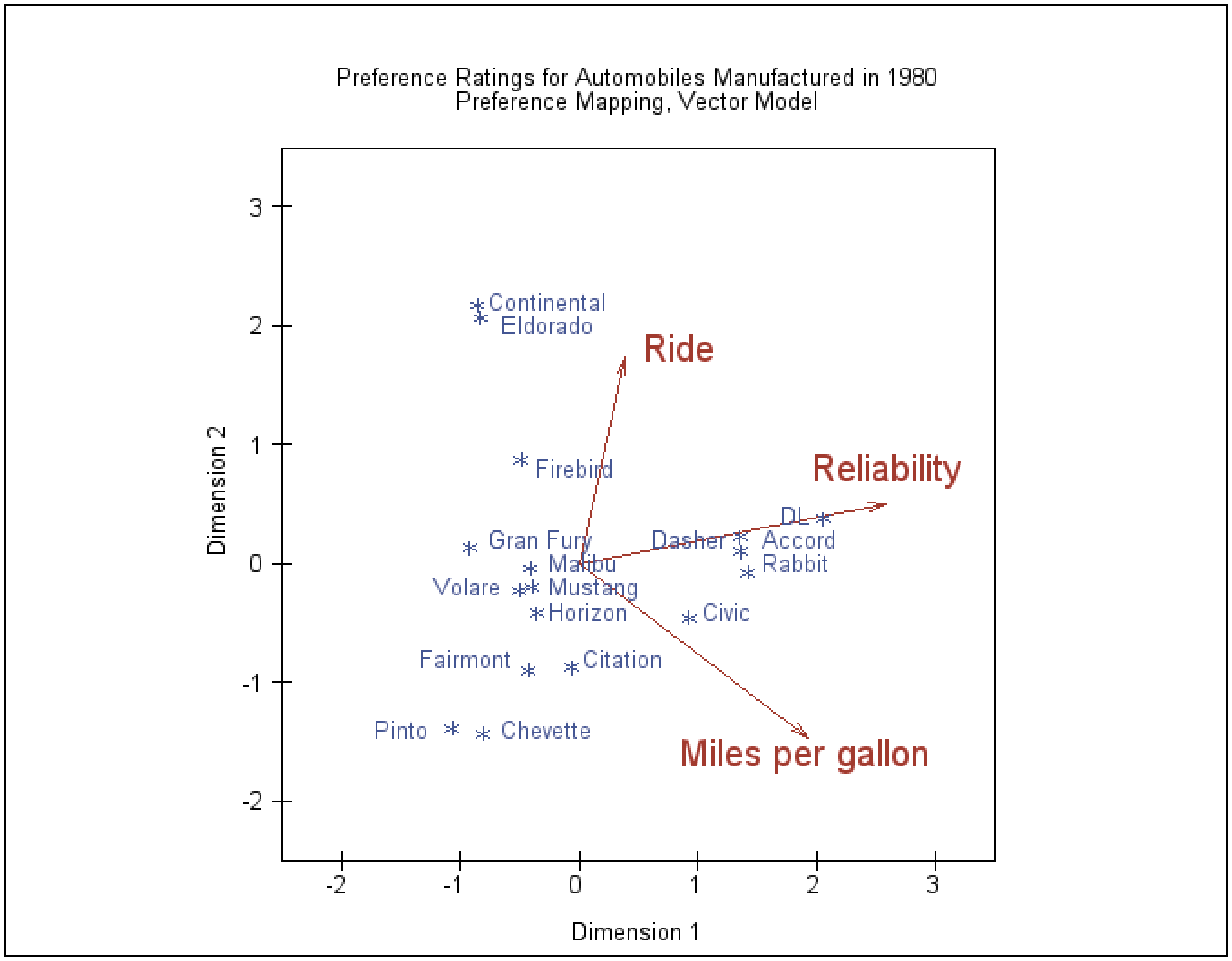


Figure 4. Preference Mapping, Vector Model

The PREFMAP Vector Model. Figure 4 contains an example in which three attribute variables (ride, reliability, and miles per gallon) are displayed in the plot of the first two principal components of the car preference data. Each of the automobiles was rated on a 1 to 5 scale, where 1 is poor and 5 is good. The end points for the attribute vectors are obtained by projecting the attribute variables into the car space. Orthogonal projections of the car points on an attribute vector give an approximate ordering of the cars on the attribute rating. The ride vector points almost straight up, indicating that the larger cars, such as the Eldorado and Continental, have the best ride. Figure 3 shows that most consumers preferred the DL, Japanese cars, and larger American cars. Figure 4 shows that the DL and Japanese cars were rated the most reliable and have the best fuel economy. The small American cars were not rated highly on any of the three dimensions.

Figure 4 is based on the simplest version of PREFMAP—the *vector model*. The vector model operates under the assumption that some is good and more is *always* better. This model is appropriate for miles per gallon and reliability—the more miles you can travel without refueling or breaking down, the better.

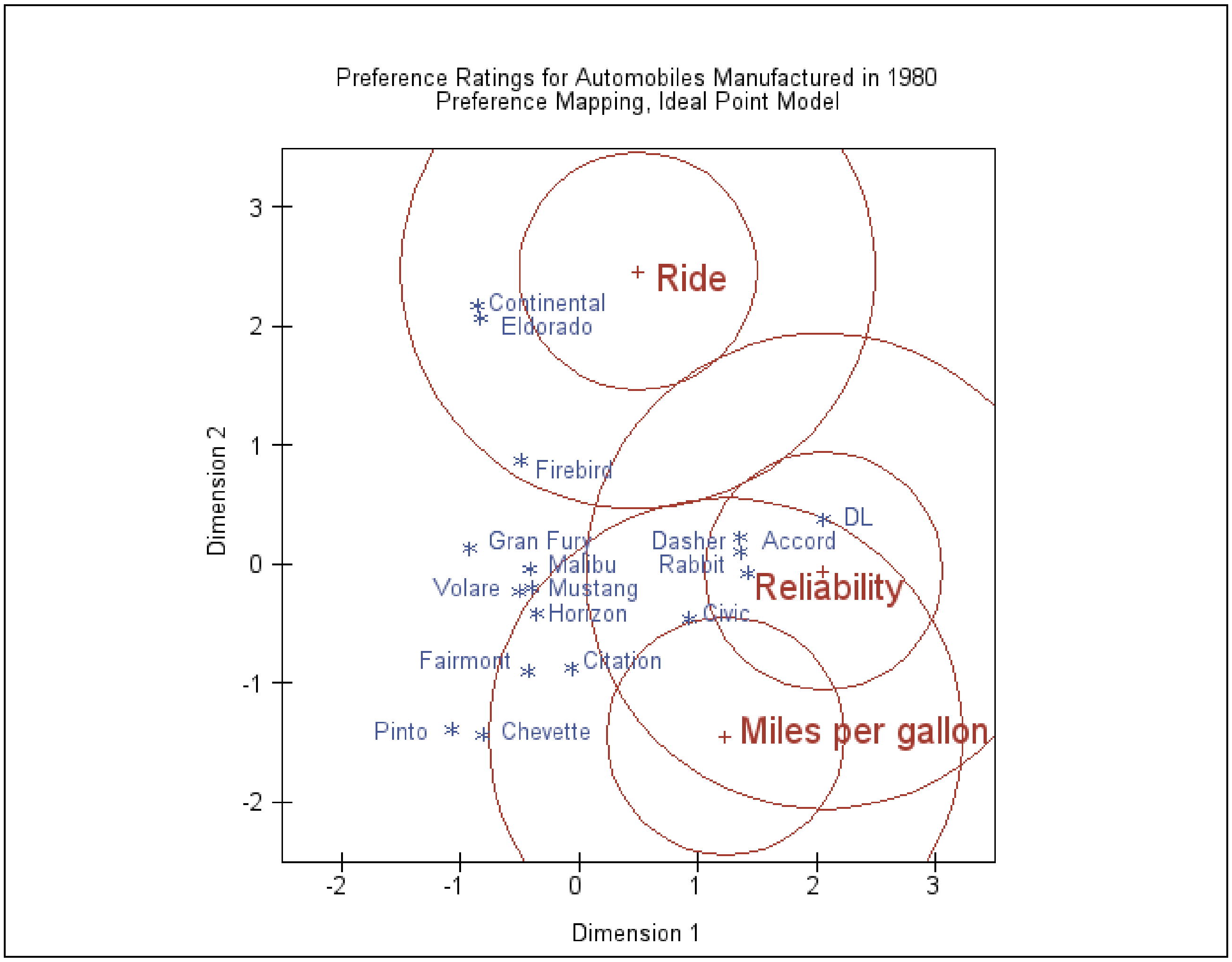


Figure 5. Preference Mapping, Ideal Point Model

The PREFMAP Ideal Point Model. The *ideal point* model differs from the vector model, in that the ideal point model does not assume that more is better, *ad infinitum*. Consider the sugar content of cake. There is an ideal amount of sugar that cake should contain—not enough sugar is not good, and too much sugar is also not good. In the cars example, the ideal number of miles per gallon and the ideal reliability are unachievable. It makes sense to consider a vector model, because the ideal point is infinitely far away. This argument is less compelling for ride; the point for a car with smooth, quiet ride may not be infinitely far away. Figure 5 shows the results of fitting an ideal point model for the three attributes. In the vector model, results are interpreted by orthogonally projecting the car points on the attribute vectors. In the ideal point model, Euclidean distances between car points and ideal points are compared. Eldorado and Continental have the best predicted ride, because they are closest to the ride ideal point. The concentric circles drawn around the ideal points help to show distances between the cars and the ideal points. The numbers of circles and their radii are arbitrary. The overall interpretations of Figures 4 and 5 are the same. All three ideal points are at the edge of the car points, which suggests the simpler vector model is sufficient for these data. The ideal point model is fit with a multiple regression model and some pre- and post-processing. The regression model uses the MDS or MDPREF coordinates as independent variables along with an additional independent variable that is the sum of squares of the coordinates. The model is a constrained *response-surface model*.

The results in Figure 5 were modified from the raw results to eliminate *anti-ideal points.* The ideal point model is a distance model. The rating data are interpreted as distances between attribute ideal points and the products. In this example, each of the automobiles was rated on these three dimensions, on a 1 to 5 scale, where 1 is poor and 5 is good. The data are the reverse of what they should be—a ride rating of 1 should mean this car is similar to a car with a good ride, and a rating of 5 should mean this car is different from a car with a good ride. So the raw coordinates must be multiplied by -1 to get ideal points. Even if the scoring had been reversed, anti-ideal points can occur. If the coefficient for the sum-of-squares variable is negative, the point is an anti-ideal point. In this example, there is the possibility of *anti-anti-ideal points.* When the coefficient for the sum-of-squares variable is negative, the two multiplications by -1 cancel, and the coordinates are ideal points. When the coefficient for the sum-of-squares variable is positive, the coordinates are multiplied by -1 to get an ideal point.

Correspondence Analysis. Correspondence analysis (CA) is used to find a low-dimensional graphical representation of the association between rows and columns of a contingency table (crosstabulation). It graphically shows relationships between the rows and columns of a table; it graphically shows the relationships that the ordinary chi-square statistic tests. Each row and column is represented by a point in a Euclidean space determined from cell frequencies. CA is a popular data analysis method in France and Japan. In France, CA analysis was developed under the strong influence of Jean-Paul Benzécri; in Japan, under Chikio Hayashi. CA is described in Lebart, Morineau, and Warwick (1984); Greenacre (1984); Nishisato (1980); Tenenhaus and Young (1985); Gifi (1990); Greenacre and Hastie (1987); and many other sources. Hoffman and Franke (1986) provide a good introductory treatment using examples from marketing research.

Questions that can be addressed with CA and MCA include: Who are my customers? Who else should be my customers? Who are my competitors' customers? Where is my product positioned relative to my competitors' products? Why is my product positioned there? How can I reposition my existing products? What new products should I create? What audience should I target for my new products?

MCA Example. Figure 6 contains a plot of the results of a multiple correspondence analysis (MCA) of a survey of car owners. The questions included origin of the car (American, Japanese, European), size of car (small, medium, large), type of car (family, sporty, work vehicle), home ownership (owns, rents), marital/family status (single, married, single and living with children, and married living with children), and sex (male, female). The variables are all categorical.

The top-right quadrant of the plot suggests that the categories single, single with kids, one income, and renting a home are associated. Proceeding clockwise, the categories sporty, small, and Japanese are associated. In the bottom-left quadrant you can see the association between being married, owning your own home, and having two incomes. Having children is associated with owning a large American family car. Such information can be used to identify target audiences for advertisements. This interpretation is based on points being located in approximately the same direction from the origin and in approximately the same region of the space. Distances between points are not interpretable in MCA.

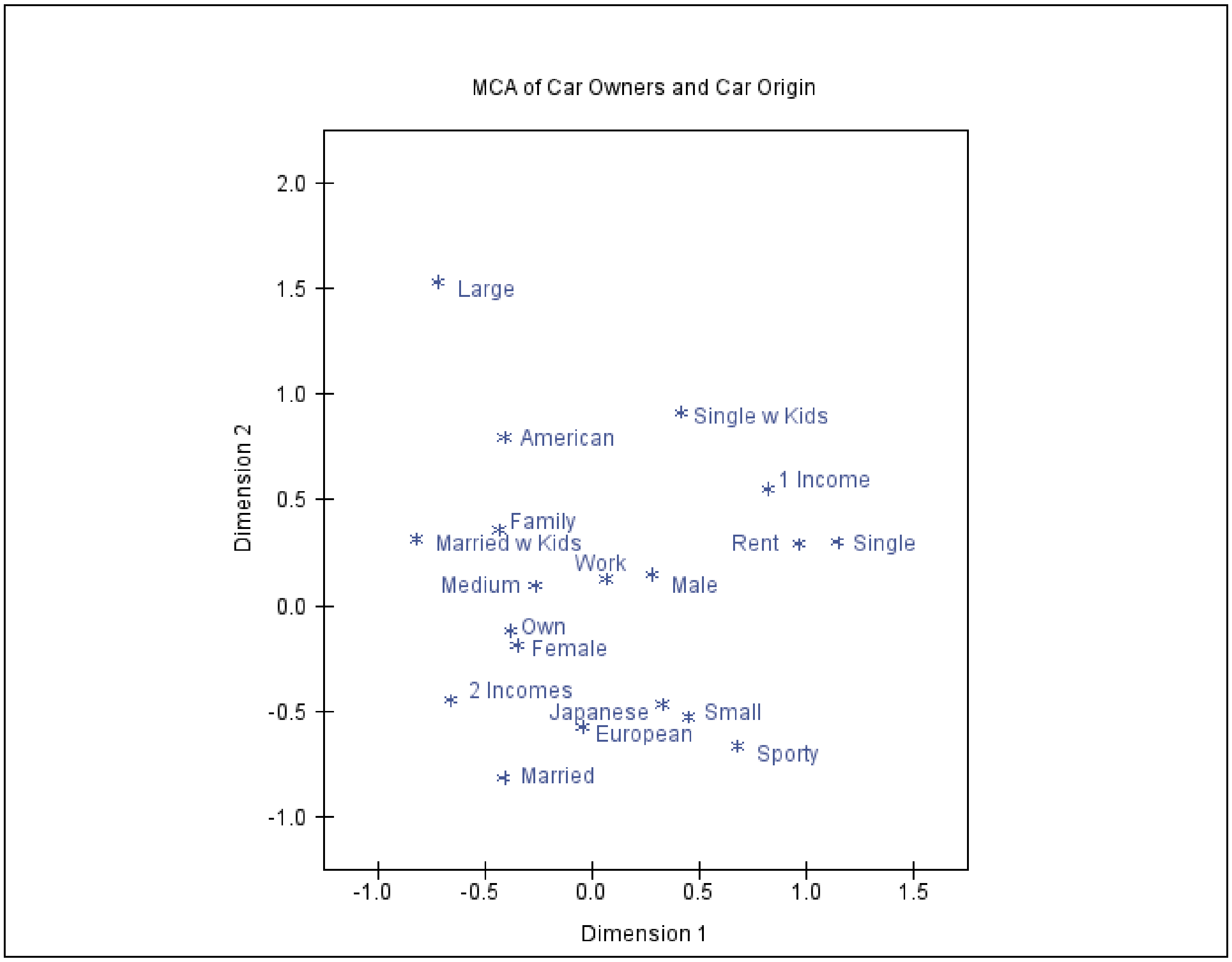


Figure 6. Multiple Correspondence Analysis

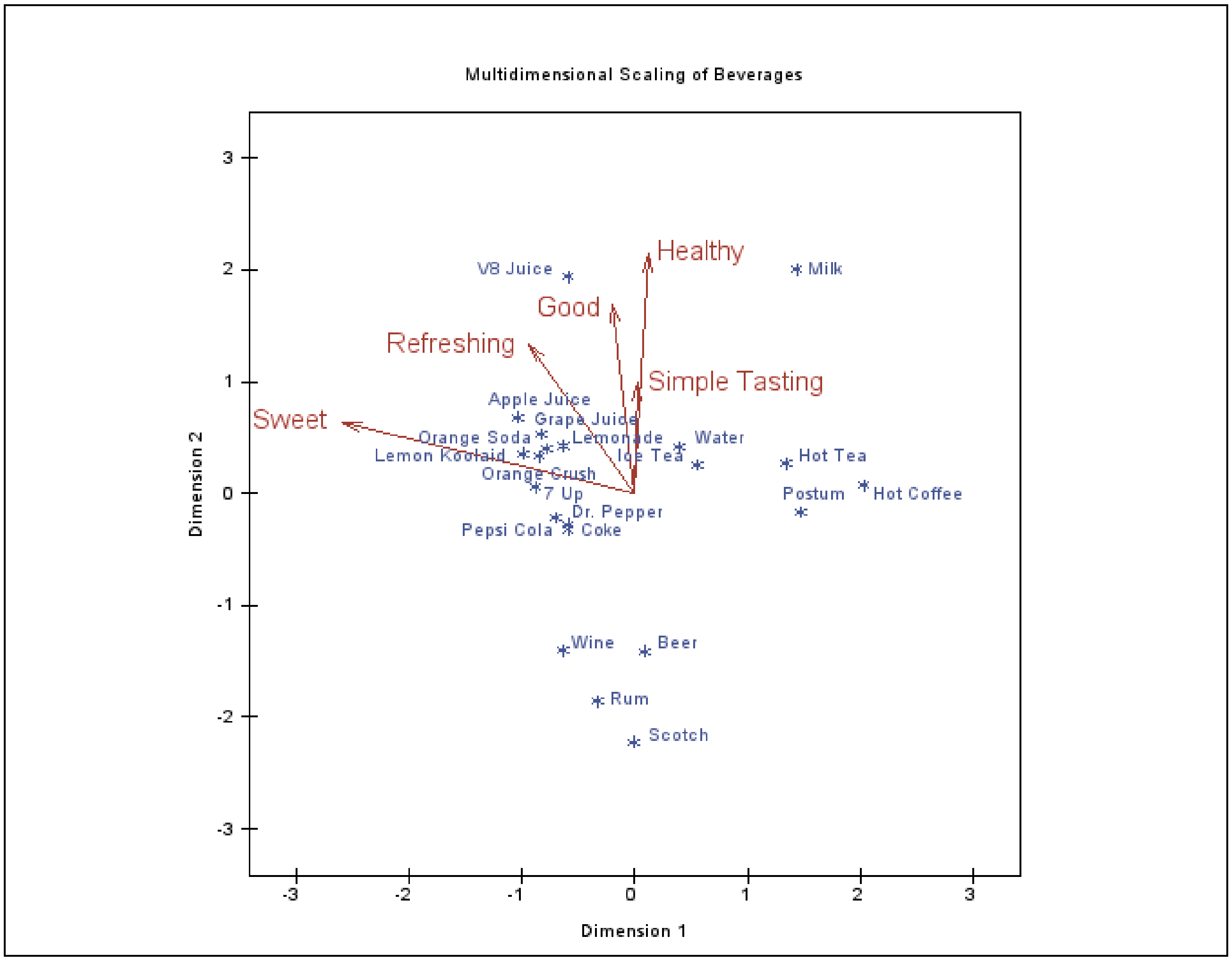


Figure 7. *MDS and PREFMAP*

Multidimensional Scaling. Multidimensional scaling (MDS) is a class of methods for estimating the coordinates of a set of objects in a space of specified dimensionality from data measuring the distances between pairs of objects (Kruskal and Wish 1978; Schiffman, Reynolds, and Young 1981; Young 1987). The data for MDS consist of one or more square symmetric or asymmetric matrices of similarities or dissimilarities between objects or stimuli. Such data are also called *proximity data*. In marketing research, the objects are often products. MDS is used to investigate product positioning.

For example, consumers were asked to rate the differences between pairs of beverages. In addition, the beverages were rated on adjectives such as Good, Sweet, Healthy, Refreshing, and Simple Tasting. Figure 7 contains a plot of the beverage configuration along with attribute vectors derived through preference mapping. The alcoholic beverages are clustered at the bottom. The juices and carbonated soft drinks are clustered at the left. Grape and Apple juice are above the carbonated and sweet soft drinks and are perceived as more healthy than the other soft drinks. Perhaps sales of these drinks would increase if they were marketed as a healthy alternative to sugary soft drinks. A future analysis, after a marketing campaign, could check to see if their positions in the plot change in the healthy direction.

Water, coffee and tea drinks form a cluster at the right. V8 Juice and Milk form two clusters of one point each. Milk and V8 are perceived as the most healthy, whereas the alcoholic beverages are perceived as least healthy. The juices and carbonated soft drinks were rated as the sweetest. Pepsi and Coke are mapped to coincident points. Postum (a coffee substitute) is near Hot Coffee, Orange Soda is near Orange Crush, and Lemon Koolaid is near Lemonade.

Geometry of the Scatter Plots. It is important that scatter plots displaying perceptual mapping information accurately portray the underlying geometry. All of the scatter plots in this chapter were created with the axes equated so that a centimeter on the y-axis represents the same data range as a centimeter on the x-axis.* *This is important.* Distances, angles between vectors, and projections are evaluated to interpret the plots. When the axes are equated, distances and angles are correctly presented in the plot. When axes are scaled independently, for example to fill the page, then the correct geometry is not presented. This important step of equating the axes is often overlooked in practice.

For MDPREF and PREFMAP, the absolute lengths of the vectors are not important since the goal is to project points on vectors, not look at scalar products of row points and column vectors. It is often necessary to change the lengths of *all* of the vectors to improve the graphical display. If all of the vectors are relatively short with end points clustered near the origin, the display will be difficult to interpret. To avoid this problem in Figure 3, *both* the x-axis and y-axis coordinates were multiplied by the same constant, 2.5, to lengthen all vectors by the same relative amount. The coordinates must *not* be scaled independently.

Conjoint Analysis

Conjoint analysis is used in marketing research to analyze consumer preferences for products and services. See Green and Rao (1971) and Green and Wind (1975) for early introductions to conjoint analysis and Green and Srinivasan (1990) for a recent review article.

Conjoint analysis grew out of the area of *conjoint measurement* in mathematical psychology. In its original form, *conjoint analysis* is a main effects analysis-of-variance problem with an ordinal scale-of-measurement dependent variable. Conjoint analysis decomposes rankings or rating-scale evaluation judgments of products into components based on qualitative attributes of the products. Attributes can include price, color, guarantee, environmental impact, and so on. A numerical *utility* or *part-worth utility* value is computed for each level of each attribute. The goal is to compute utilities such that the rank ordering of the sums of each product's set of utilities is the same as the original rank ordering or violates that ordering as little as possible.

When a monotonic transformation of the judgments is requested, a *nonmetric conjoint analysis* is performed. Nonmetric conjoint analysis models are fit iteratively. When the judgments are not transformed, a *metric conjoint analysis* is performed. Metric conjoint analysis models are fit directly with ordinary least squares. When all of the attributes are nominal, the metric conjoint analysis problem is a simple main-effects ANOVA model. The attributes are the independent variables, the judgments comprise the dependent variable, and the utilities are the parameter estimates from the ANOVA model. The metric conjoint analysis model is more restrictive than the nonmetric model and will generally fit the data less well than the nonmetric model. However, this is not necessarily a disadvantage since over-fitting is less of a problem and the results should be more reproducible with the metric model.

*If the plot axes are not equated in this chapter, it is due to unequal distortions of the axes that occurred during the final formatting or printing process.

In both metric and nonmetric conjoint analysis, the respondents are typically not asked to rate all possible combinations of the attributes. For example, with five attributes, three with three levels and two with two levels, there are $3 \times 3 \times 3 \times 2 \times 2 = 108$ possible combinations. Rating that many combinations would be difficult for consumers, so typically only a small fraction of the combinations are rated. It is still possible to compute utilities, even if not all combinations are rated. Typically, combinations are chosen from an *orthogonal array* which is a *fractional-factorial design*. In an orthogonal array, the zero/one indicator variables are uncorrelated for all pairs in which the two indicator variables are not from the same factor. The main effects are orthogonal but are confounded with interactions. These interaction effects are typically assumed to be zero.

Questions that can be addressed with conjoint analysis include: How can I reposition my existing products? What new products should I create? What audience should I target for my new products?

Consider an example in which the effects of four attributes of tea on preference were evaluated. The attributes are temperature (Hot, Warm, and Iced), sweetness (No Sugar, 1 Teaspoon, 2 Teaspoons), strength (Strong, Moderate, Weak), and lemon (With Lemon, No Lemon). There are four factors: three with three levels and one with two levels. Figure 8 contains the results.[*]

Sweetness was the most important attribute (the importance is 55.795). This consumer preferred two teaspoons of sugar over one teaspoon, and some sugar was preferred over no sugar. The second most important attribute was strength (25.067), with moderate and strong tea preferred over weak tea. This consumer's most preferred temperature was iced, and no lemon was preferred over lemon.

Software

SAS includes software that implements these methods. SAS/STAT software was used to perform the analyses for all of the examples. Perceptual mapping methods are described with more mathematical detail starting on page 1263.

Correspondence Analysis. The SAS/STAT procedure CORRESP performs simple and multiple correspondence analysis and outputs the coordinates for plotting. Raw data or tables may be input. Supplementary classes are allowed.

Multidimensional Preference Analysis. The SAS/STAT procedure PRINQUAL performs multidimensional preference analysis and outputs the coordinates for plotting. Nonmetric MDPREF, with transformations of continuous and categorical variables, is also available.

Preference Mapping. The SAS/STAT procedure TRANSREG performs preference mapping and outputs the coordinates. Nonmetric PREFMAP, with transformations of continuous and categorical variables, is also available.

Multidimensional Scaling. The SAS/STAT procedure MDS performs multidimensional scaling and outputs the coordinates. Metric, nonmetric, two-way, and three-way models are available.

[*]See page 681 for more information about conjoint analysis. Note that the results in Figure 8 have been customized using ODS. See page 683 for more information about customizing conjoint analysis output.

```
                    Conjoint Analysis of Tea-Tasting Data

                           The TRANSREG Procedure

          The TRANSREG Procedure Hypothesis Tests for Linear(subj2)


         Univariate ANOVA Table Based on the Usual Degrees of Freedom

                                   Sum of          Mean
     Source               DF      Squares        Square     F Value     Pr > F

     Model                 7     617.7222      88.24603       32.95     <.0001
     Error                10      26.7778       2.67778
     Corrected Total      17     644.5000


               Root MSE            1.63639     R-Square     0.9585
               Dependent Mean     12.16667     Adj R-Sq     0.9294
               Coeff Var          13.44979


            Utilities Table Based on the Usual Degrees of Freedom

                                                        Importance
                                              Standard  (% Utility
            Label                    Utility     Error      Range)

            Intercept                12.1667   0.38570

            Lemon: No                 0.7222   0.38570       7.008
            Lemon: Yes               -0.7222   0.38570

            Temperature: Hot          0.5000   0.54546      12.129
            Temperature: Iced         1.0000   0.54546
            Temperature: Warm        -1.5000   0.54546

            Sweetness: No Sugar      -7.3333   0.54546      55.795
            Sweetness: 1 Teaspoon     3.1667   0.54546
            Sweetness: 2 Teaspoons    4.1667   0.54546

            Strength: Moderate        1.8333   0.54546      25.067
            Strength: Strong          1.5000   0.54546
            Strength: Weak           -3.3333   0.54546
```

Figure 8. Conjoint Analysis

Scatter Plots. The Base SAS procedure PLOT can plot the results from these analyses and optimally position labels in the scatter plot. PROC PLOT uses an algorithm, developed by Kuhfeld (1991), that uses a heuristic approach to avoid label collisions. Labels up to 200 characters long can be plotted.

The `%PlotIt` macro, was used to create graphical scatter plots of labeled points. There are options to draw vectors to certain symbols and draw circles around other symbols. This macro is in the SAS autocall macro library. See page 1231. With the 9.2 SAS release, the `%PlotIt` macro is much less necessary than it was in previous releases. The graphical displays for CA, MCA, PREFMAP, MDPREF, and MDS are now automatically created through ODS Graphics.

Conjoint Analysis. The SAS/STAT procedure TRANSREG can perform both metric and nonmetric conjoint analysis. PROC TRANSREG can handle both *holdout* observations and *simulations*. Holdouts are ranked by the consumers but are excluded from contributing to the analysis. They are used to validate the results of the study. Simulation observations are not rated by the consumers and do not contribute to the analysis. They are scored as passive observations. Simulations are *what-if* combinations. They are combinations that are entered to get a prediction of what their utility would have been if they had been rated. Conjoint analysis is described in more detail starting on page 681.

The `%MktEx` macro can generate orthogonal designs for both main-effects models and models with interactions. Nonorthogonal designs—for example, when strictly orthogonal designs require too many observations—can also be generated. Nonorthogonal designs can be used in conjoint analysis studies to minimize the number of stimuli when there are many attributes and levels. This macro is in the SAS autocall macro library and is also available free of charge on the Web: `http://support.sas.com/resources/papers/tnote/tnote_marketresearch.html`. Experimental design and the `%MktEx` macro are described in more detail in starting on pages 53, 243, 265, 285, 681, 803, and 1017.

Other Data Analysis Methods. Other procedures that are useful for marketing research include the SAS/STAT procedures for regression, ANOVA, discriminant analysis, principal component analysis, factor analysis, categorical data analysis, covariance analysis (structural equation models), and the SAS/ETS procedures for econometrics, time series, and forcasting. Discrete choice data can be analyzed with multinomial logit models using the PHREG procedure. Discrete choice is described in more detail in starting on page 285.

Conclusions

Marketing research helps you understand your customers and your competition. Correspondence analysis compactly displays survey data to aid in determining what kinds of consumers are buying your products. Multidimensional preference analysis and multidimensional scaling show product positioning, group preferences, and individual preferences. Plots from these methods may suggest how to reposition your product to appeal to a broader audience. They may also suggest new groups of customers to target. Preference mapping is used as an aid in understanding MDPREF and MDS results. PREFMAP displays product attributes in the same plot as the products. Conjoint analysis is used to investigate how consumers trade off product attributes when making a purchasing decision.

The insight gained from perceptual mapping and conjoint analysis can be a valuable asset in marketing decision making. These techniques can help you gain insight into your products, your customers, and your competition. They can give you the edge in gaining a competitive advantage.

Introducing the Market Research Analysis Application

Wayne E. Watson

Abstract

Market research focuses on assessing the preferences and choices of consumers and potential consumers. A new component of SAS/STAT software in Release 6.11 of the SAS System is an application written in SAS/AF that provides statistical and graphical techniques for market research data analysis. The application allows you to employ statistical methods such as conjoint analysis, discrete choice analysis, correspondence analysis, and multidimensional scaling through intuitive point-and-click actions.*

Conjoint Analysis

Conjoint analysis is used to evaluate consumer preference. If products are considered to be composed of attributes, conjoint analysis can be used to determine what attributes are important to product preference and what combinations of attribute levels are most preferred.

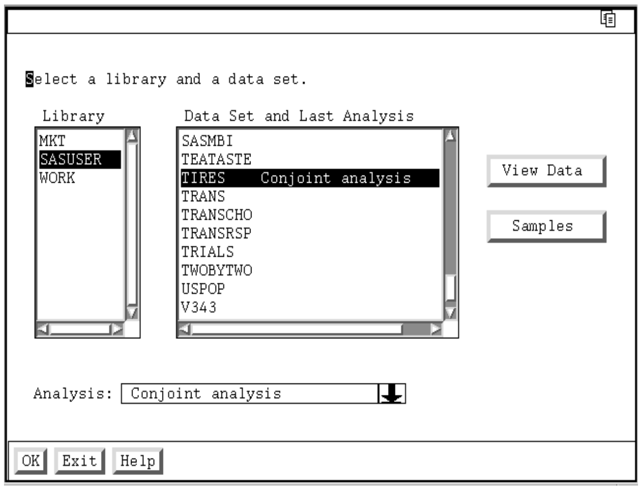
Usually, conjoint analysis is a main-effects analysis of variance of ordinally-scaled dependent variables. Preferences are used as dependent variables, and attributes are used as independent variables. Often, a monotone transformation is used with the dependent variables to fit a model with no interactions.

As an example, suppose you have four attributes that you think are related to automobile tire purchase. You want to know how important each attribute is to consumers' stated preferences for a potential tire purchase. The four attributes under investigation are

- brand name
- expected tread mileage
- purchase price
- installation cost

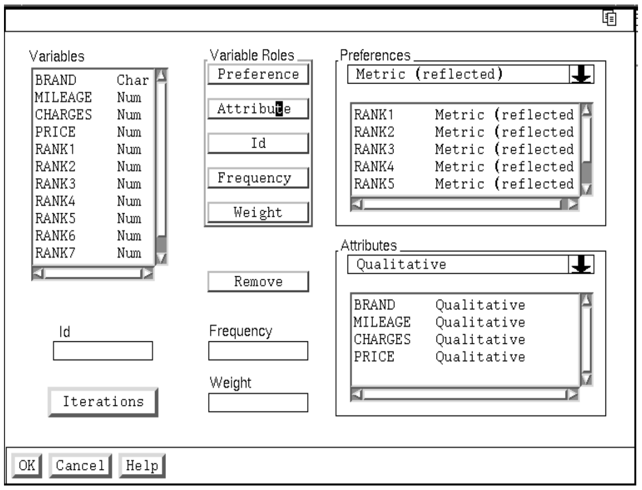
The attributes of brand name, tread mileage, and purchase price have three possible values and installation cost has two values. The values for each attribute are:

*For current documentation on the Market Research Application see SAS Institute Inc, *Getting Started with The Market Research Application*, Cary, NC: SAS Institute Inc., 1997, 56 pp. This paper was written and presented at SUGI 20 (1995) by Wayne E. Watson. This paper was also presented to SUGI-Korea (1995) by Warren F. Kuhfeld. Wayne Watson is a Research Statistician at SAS and wrote the Marketing Research Application which uses procedures and macros written by Warren F. Kuhfeld. Copies of this chapter (MR-2010B), the other chapters, sample code, and all of the macros are available on the Web `http://support.sas.com/resources/papers/tnote/tnote_marketresearch.html`. For help, please contact SAS Technical Support. See page 25 for more information.



```
Select a library and a data set.

Library      Data Set and Last Analysis
MKT          SASMBI
SASUSER      TEATASTE
WORK         TIRES     Conjoint analysis
             TRANS
             TRANSCHO
             TRANSRSP
             TRIALS
             TWOBYTWO
             USPOP
             V343

Analysis: Conjoint analysis
```

Figure 1. Selecting a Data Set and Analysis



```
Variables          Preferences
BRAND     Char     Metric (reflected)
MILEAGE   Num      RANK1   Metric (reflected
CHARGES   Num      RANK2   Metric (reflected
PRICE     Num      RANK3   Metric (reflected
RANK1     Num      RANK4   Metric (reflected
RANK2     Num      RANK5   Metric (reflected
RANK3     Num
RANK4     Num      Attributes
RANK5     Num      Qualitative
RANK6     Num      BRAND     Qualitative
RANK7     Num      MILEAGE   Qualitative
                   CHARGES   Qualitative
                   PRICE     Qualitative
```

Figure 2. Conjoint Analysis Variable Selection

```
Brand:             Michelin, Goodyear, Firestone
Tread Mileage:     40,000, 60,000, 80,000
Price:             $45.00, $60.00, $75.00
Installation Cost: $0.00, $7.50
```

Seven respondents are asked to rank in order of preference 18 out of the possible 54 combinations. Although rankings are used in this example, preference ratings are frequently used in conjoint analysis.

Invoking the Application. With the data in the SAS data set, SASUSER.TIRES, you can invoke the Market Research application and perform a conjoint analysis. The application is invoked by issuing the "market" command on any command line.

Selecting a Data Set and Analysis. The first window displayed requires you to select a data set and an analysis. Because your data set is SASUSER.TIRES, select SASUSER as the library in the left-hand list box and TIRES as the data set in the right-hand list box. Then, select an analysis by clicking on the down arrow to the right of the analysis name field below the list boxes and select "Conjoint Analysis" from the displayed popup menu. See Figure 1.

View the data by pressing the View Data button and then selecting "Data values." The other selection under the View Data button,"Variable attributes," displays information about each variable.

Selecting Variables. To proceed with the analysis once you have selected a data set and an analysis, press the OK button at the bottom of the window.

The analysis requires preference and attribute variables. The preference variables are the ranks from the seven respondents and the attribute variables are the four factors. See Figure 2.

You can choose to perform a metric or a non-metric conjoint analysis; the metric analysis uses the ranks as they are, while the non-metric analysis performs a monotone transformation on the ranks. To set the measurement type for the preferences, click on the down arrow in the Preferences box at the top right of the window. Select "Metric (reflected)." "Reflected" is used because the lowest rank value, 1, corresponds to the most preferred offering. If the highest preference value corresponded to the most preferred offering, the "Metric" selection should be used instead.

To select preference variables, select RANK1, RANK2, ... RANK7 in the Variables list box on the left side of the window, and press the Preference button in the Variable Roles box.

Likewise, you must select a measurement type for the attribute variables you want to use. The default measurement type for attributes is Qualitative, which treats the variable as a set of dummy variables with the coefficients of the dummy variables summing to 0. In this way, the utility coefficients [*] of each attribute sum to 0.

Use this measurement type for all four attribute variables, BRAND, MILEAGE, CHARGES, and PRICE. After selecting these four variables in the Variables list box, press the Attribute button in the Variable Roles box. Alternatively, you could use the "Continuous" measurement type for MILEAGE, CHARGES, or PRICE because these attributes are quantitative in nature.

To delete one or more of the Preference or Attribute variables, either double-click on each one in the appropriate right-hand list box or select them in any of the three list boxes and press the Remove button.

To obtain help about the window, press the Help button at the bottom of the window or click on any of the border titles on the window, for example, "Variables," "Variable Roles," "Preferences."

Once the variables have been selected, press the OK button at the bottom of the window to perform the analysis. To change the analysis, return to the Variable Selection window by pressing the Variables button on the analysis main window.

Results. The first result is a plot of the relative importance of each attribute. Relative importance is a measure of importance of the contribution of each attribute to overall preference; it is calculated by dividing the range of utilities for each attribute by the sum of all ranges and multiplying by 100.

In the example, Tire Mileage is the most important attribute with an average relative importance of 49%. The box-and-whisker plot displays the first and third quartiles as the ends of the box, the maximum and minimum as the whiskers (if they fall outside the box), and the median as a vertical bar in the interior of each box. See Figure 3.

To display a selection of additional results, press the Results button on the window. The first selection, the Utilities Table window, displays the utility coefficients for each level of an attribute for all preferences (the dependent variables). The relative importance of each attribute is displayed separately for each preference variable. This table illustrates that BRAND is the most important attribute for RANK1, the first respondent, and Michelin is the most preferred brand, because it has the highest utility coefficient value. Thus, the first respondent preferred a 80,000 mile, $45 Michelin with no installation charge.

After closing this window, you can view these results in graphical form by pressing the Results button again and selecting "Utilities plots." The plot of the Brand utilities indicates that one respondent clearly prefers Michelin while the other respondents only mildly prefer one brand over another.

To change the plot from the BRAND to the MILEAGE attribute, select MILEAGE in the list box at the right. All but one person prefer longer over shorter mileage tires, and that one prefers the 60,000 mile tire. You can examine plots for the PRICE and CHARGES attributes in the same way.

[*]Utility coefficients are estimates of the value or worth to a subject of each level of an attribute. The most preferred combination of attributes for a subject is the one with the attribute levels having the highest utility coefficient values for each attribute.

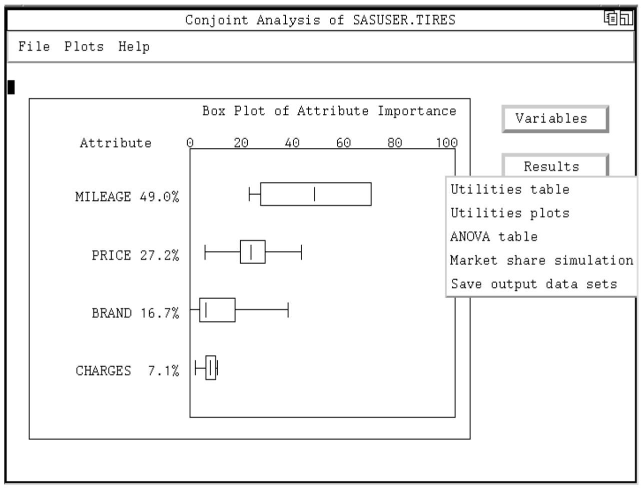
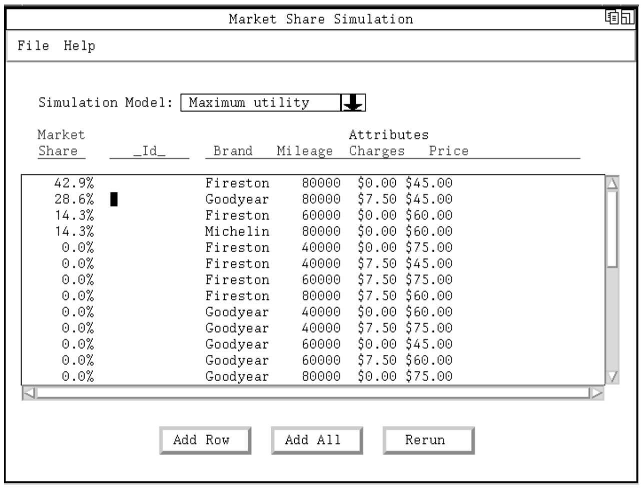


Figure 3. Plot of Relative Importance of Attributes



Simulation Model: Maximum utility

Market Share	_Id_	Brand	Mileage	Charges	Price
42.9%		Fireston	80000	$0.00	$45.00
28.6%		Goodyear	80000	$7.50	$45.00
14.3%		Fireston	60000	$0.00	$60.00
14.3%		Michelin	80000	$0.00	$60.00
0.0%		Fireston	40000	$0.00	$75.00
0.0%		Fireston	40000	$7.50	$45.00
0.0%		Fireston	60000	$7.50	$75.00
0.0%		Fireston	80000	$7.50	$60.00
0.0%		Goodyear	40000	$0.00	$60.00
0.0%		Goodyear	40000	$7.50	$75.00
0.0%		Goodyear	60000	$0.00	$45.00
0.0%		Goodyear	60000	$7.50	$60.00
0.0%		Goodyear	80000	$0.00	$75.00

Figure 4. Estimating Market Share

Estimating Market Share. You also can calculate the expected market share for each tire purchase alternative in the sample. To do so, press the Results button and select “Market Share Simulation.” The entry in the table with the largest market share is the 80,000 mile, $45 Firestone with no installation charge. It is expected to account for 42.9% of the market. The maximum utility simulation model, the default, was used to calculate the market share. You can choose from two other models: the logit model and the Bradley-Terry-Luce model. Click on the down arrow at the top of the window and select the desired model from the displayed list. See Figure 4.

Only 18 of the 54 possible tire purchase combinations were presented to the respondents. You may want to predict the expected market share of one or more of the combinations that were not present in the sample. To do so, press the Add Row button at the bottom of the window and fill in the observation in the top row of the table. Click on “-Select-” in each attribute column and select the desired level. If the observation that you create is a duplicate, a warning message is displayed. You can modify the contents of the Id column to contain a description of your own choice. After you have added some combinations, you can produce the expected market shares by pressing the Rerun button.

As an example an 80,000 mile, $45 Michelin with no installation charges would be expected to have a 64.3% market share if it was the only combination added to the original sample. Adding combinations may change the estimated market share of the other combinations.

Discrete Choice Analysis

Conjoint analysis is used to examine the preferences of consumers. The rationale for the use of preferences is that they indicate what people will choose to buy. Often in market research, the choices that consumers actually make are the behavior of interest. In these instances, it is appropriate to analyze choices directly using discrete choice analysis.

In discrete choice analysis, the respondent is presented with several choices and selects one of them. As in conjoint analysis, the factors that define the choice possibilities are called attributes. Here, they are called choice attributes to distinguish them from other factors, like demographic variables, that may be of interest but do not contribute to the definition of the choices. Each set of possible choices is called a choice set.

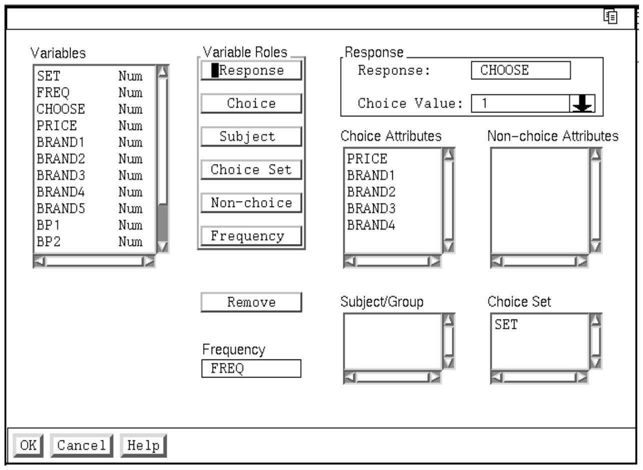


Figure 5. Discrete Choice Analysis Variable Selection

This example has choice possibilities defined by two attributes, price and brand. Five choice alternatives are presented at a time to a respondent, from which one alternative is chosen. Eight of these choice sets are presented, each one with a different set of five combinations of price and brand.

To change to a different data set or analysis, select "File → New dataset/analysis" on the main analysis window. Each time you change the data set or analysis or exit the application, you are asked if you want save the changes that you have made during the session. On the data set selection window, select the PRICE data set in the SASUSER library and then select "Discrete choice analysis." To continue, press the OK button.

With the other analyses in the application, you would be taken directly to the appropriate variable selection window. With discrete choice analysis, a supplementary window is displayed to help you determine if your data are in the appropriate form.

With discrete choice analysis, the structure of the data is important and must be in one of several layouts. After specifying if your data are contained in one or two data sets and whether a frequency variable is used, you can view the appropriate layout by pressing the Examine button. The most important requirement of the data layout is that all choice alternatives must be included, whether chosen or not.

If your data are not in the proper form, they must be rearranged before proceeding with the analysis. If your data are in the proper form, continue with the analysis by pressing the OK button. If not, press the Cancel button.

On the Variable Selection window that appears next, you must select several required variables: a response variable, some choice attribute variables, and a subject variable. Optionally, you can also choose a frequency variable and some non-choice attribute variables. If you select a frequency variable, a subject variable is not necessary.

For this example, select CHOOSE as the response variable. You also must indicate which value of the variable represents a choice. Click on the down arrow to the right of "Choice Value:" and select 1 from the list. In this example the value 1 indicates the chosen alternative and the value 0 indicates the non-chosen alternatives. See Figure 5.

Next, select PRICE and BRAND1, BRAND2, ..., BRAND4 as Choice attributes. BRAND is a nominal variable with five levels. It can be represented as four dummy-coded variables. [*]

Select FREQ as the frequency variable. The frequency variable contains the count of the number of times that a choice alternative was selected.

Because the data include more than one choice set, a Choice Set variable is needed; the choice set variable in this example is SET. After selecting the appropriate variables, press the OK button to perform the analysis.

On the analysis main window, a bar chart is displayed of the significances of each of the choice and non-choice attributes. The chart illustrates that PRICE, BRAND1, BRAND2, and BRAND4 are significant.

You can view other results by pressing the Results button and selecting “Statistics,” “Choice probabilities,” or “Residual plots” from the ensuing menu. Overall model fit statistics and parameter estimates for the attributes are available from the Statistics window. Probabilities for each choice alternative are available from the Choice Probabilities window. Plots of residual and predicted values are available from the Residual Plots window.

Correspondence Analysis

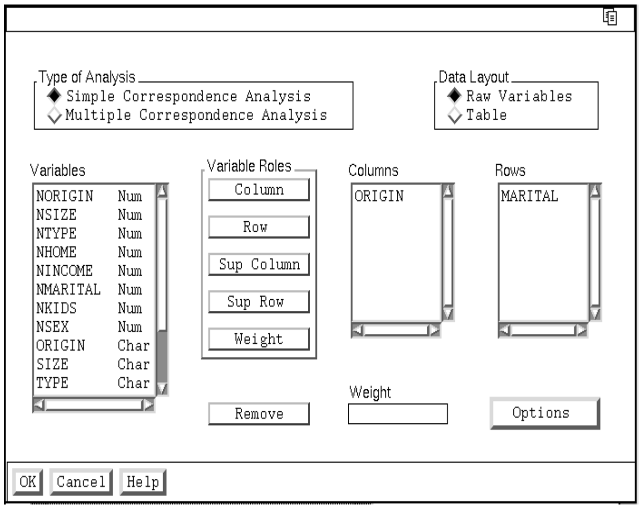
Categorical data are frequently encountered in the field of market research. Correspondence analysis is a technique that graphically displays relationships among the rows and columns in a contingency table. In the resulting plot there is a point for each row and each column of the table. Rows with similar patterns of counts have points that are close together, and columns with similar patterns of counts have points that are close together.

The CARS data set in the SASUSER library is used as an example (also described in the *SAS/STAT User’s Guide*). The CARS data are a sample of individuals who were asked to provide information about themselves and their cars. The pertinent questions for the example are country of origin of their car and their family status.

Simple Correspondence Analysis. Simple correspondence analysis analyzes a contingency table made up of one or more column variables and one or more row variables. To select a data set on which to perform a correspondence analysis, select “File → New dataset/analysis” on the main analysis window. First, select the CARS data set, then select “Correspondence analysis” as the analysis, and then press the OK button.

This example uses raw variables instead of an existing table. The desired type of analysis (simple correspondence analysis) and data layout (raw variables) are default selections on the Variable Selection window. Select ORIGIN, the country of origin of the car, as the column variable and MARITAL, family status, as the row variable to create the desired contingency table. See Figure 6.

[*]Each dummy-coded variable has the value of 1 for a different level of the attribute. In this way, each dummy-coded variable represents the presence of that level and the absence of the other levels.



```
NORIGIN   Num
NSIZE     Num
NTYPE     Num
NHOME     Num
NINCOME   Num
NMARITAL  Num
NKIDS     Num
NSEX      Num
ORIGIN    Char
SIZE      Char
TYPE      Char
```

Figure 6. Simple Correspondence Analysis Variable Selection

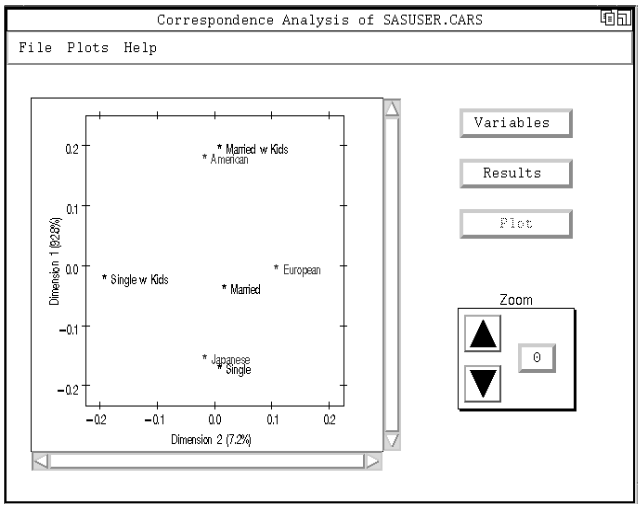


Figure 7. Correspondence Analysis Plot

Plot. The plot displays the column points and row points. The first example in the *SAS/STAT User's Guide* provides an interpretation of the plot. The interpretation has two aspects: what each dimension represents and what the relationship of the points in the dimensional space represents. An interpretation of the vertical dimension is that it represents the country of origin of the cars, with most of the influence coming from whether the car is American or Japanese. The horizontal dimension appears to represent "Single with kids" versus all of the other values. See Figure 7.

Although the row and column points are spread throughout the plot, "married" and "single" appear to be slightly more similar to each other than any of the other points. Keep in mind that distances between row and column points cannot be compared, only distances among row points and distances among column points. However, by treating the country-of-origin points as lines drawn from the 0,0 point and extending off the graph, you can see that the "Married with kids" point is closest to the American car line and the "Single" point is closest to the Japanese car line.

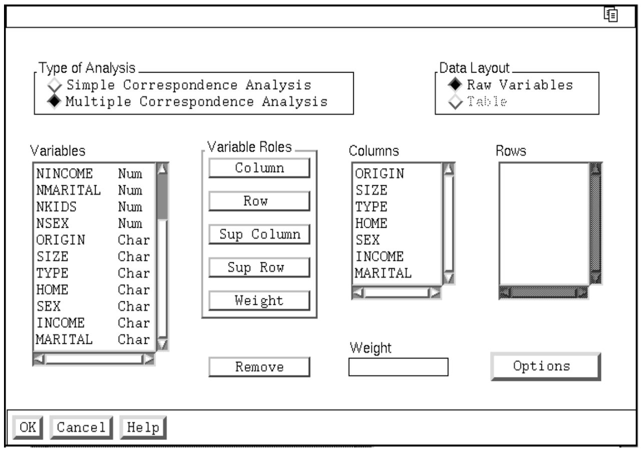
Plot Controls. To enlarge the plot, click on the up arrow in the zoom control box. To return the plot to its zero zoom state, click on the [0] button. If the plot is zoomed, you can move the plot left and right and up and down using the scroll bars.

Results. You can view other results by pressing the Results button and selecting "Inertia table," "Statistics," or "Frequencies." The Inertia Table window lists the singular values and inertias for all possible dimensions in the analysis. The Statistics window displays tables of statistics that aid in the interpretations of the dimensions and the points: the row and column coordinates, the partial contributions to inertia, and the squared cosines. The Frequency Table window displays observed, expected, and deviation contingency tables and row and column profiles.

Multiple Correspondence Analysis. In a multiple correspondence analysis, only column variables are used. They are used to create a Burt table [*] which is then used in the analysis.

The same data set can be used to illustrate multiple correspondence analysis. Return to the Variables Selection window by pressing the Variables button on the main analysis window. See Figure 8. Perform the following steps:

1. Remove the current column and row variables either by double-clicking on them or by selecting them and pressing the Remove button.
2. Select "Multiple Correspondence Analysis" in the Type of Analysis box in the upper left of the window.
3. Select the column variables ORIGIN, TYPE, SIZE, HOME, SEX, INCOME, and MARITAL by clicking on the ORIGIN variable and dragging through the list to the MARITAL variable, then press the Column button.
4. Press the OK button to perform the analysis.


Type of Analysis
◇ Simple Correspondence Analysis
◆ Multiple Correspondence Analysis

Data Layout
◆ Raw Variables
◇ Table

Variables

NINCOME	Num
NMARITAL	Num
NKIDS	Num
NSEX	Num
ORIGIN	Char
SIZE	Char
TYPE	Char
HOME	Char
SEX	Char
INCOME	Char
MARITAL	Char

Columns

ORIGIN
SIZE
TYPE
HOME
SEX
INCOME
MARITAL

Rows

Weight

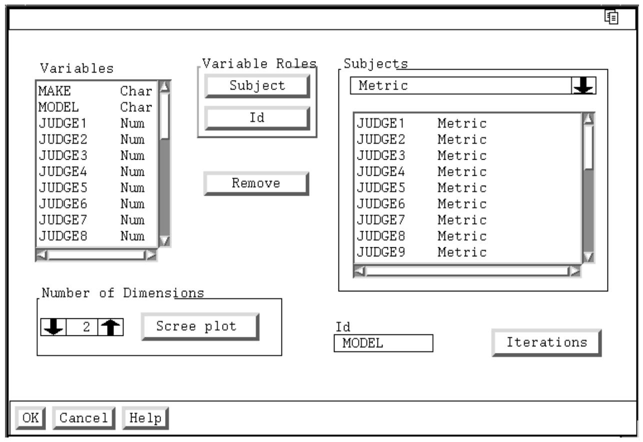
Figure 8. Multiple Correspondence Analysis Variable Selection

The distances between points of different variables can be interpreted in multiple correspondence analysis because they are all column points. However, the multiple correspondence analysis example has more dimensions (12) to interpret and examine than the single correspondence analysis example (2). The total number of dimensions can be examined in the inertia table, which is accessed from the Results button.

By default, a two-dimensional solution is computed. To request a higher dimensional solution, open the Variable Selection window, press the Options button, and select (or enter) the desired number of dimensions.

If you request a three-dimensional (or higher) solution, you can plot the dimensions two at a time by pressing the Plot button and selecting dimensions for the x axis and the y axis.

[*] A Burt table is a partitioned symmetric matrix containing all pairs of crosstabulations among a set of categorical variables. For further explanation, see the *SAS/STAT User's Guide*



```
Variables
MAKE      Char
MODEL     Char
JUDGE1    Num
JUDGE2    Num
JUDGE3    Num
JUDGE4    Num
JUDGE5    Num
JUDGE6    Num
JUDGE7    Num
JUDGE8    Num

Subjects
Metric
JUDGE1    Metric
JUDGE2    Metric
JUDGE3    Metric
JUDGE4    Metric
JUDGE5    Metric
JUDGE6    Metric
JUDGE7    Metric
JUDGE8    Metric
JUDGE9    Metric

Number of Dimensions  2
Id  MODEL
```

Figure 9. MDPREF Analysis Variable Selection

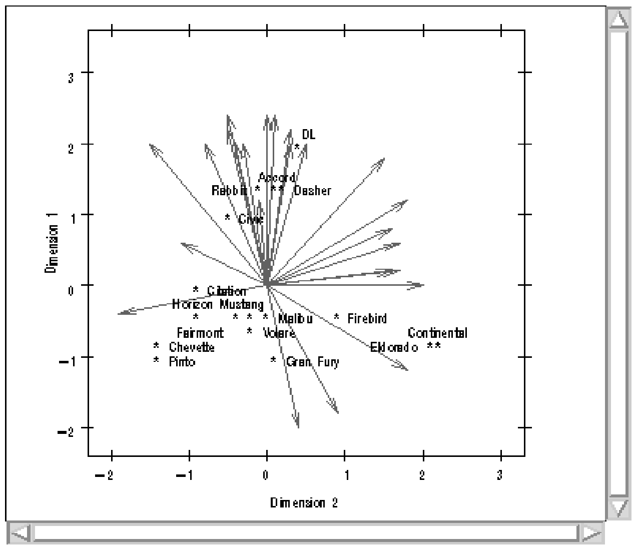


Figure 10. MDPREF Plot

Multidimensional Preference Analysis

With conjoint analysis, respondents indicate their preferences for products that are composed of attributes determined by the experimenter. Sometimes, the data of interest may be preferences of existing products for which relevant attributes are not defined for the respondent. Multidimensional preference analysis (MDPREF) is used to analyze such data.

MDPREF is a principal component analysis of a data matrix whose columns correspond to people and whose rows correspond to objects, the transpose of the usual people by objects multivariate data matrix.

The CARPREF data set in the SASUSER library is used as an example (also described in the *SAS/STAT User's Guide*. It contains data about the preferences of 25 respondents for 17 cars. The preferences are on a scale of 0 to 9 with 0 meaning a very weak preference and 9 meaning a very strong preference. Select the data set and analysis as described in the preceding examples.

As in conjoint analysis, you can choose to perform a metric or non-metric analysis. Choose the measurement type by clicking the arrow in the upper right corner of the window and selecting the desired type. Other, less frequently used, types are available under the "Other" selection. The measurement type is used for all subsequently selected Subject variables. Infrequently, subject variables with different types may be used.

For the example, use the Metric measurement type. Select the preference ratings of each respondent, JUDGE1, JUDGE2, ..., JUDGE25, as Subject variables. Also, select MODEL as the Id variable. See Figure 9.

You also can set the number of dimensions for the analysis; the default is two. A scree plot of the eigenvalues is useful in determining an appropriate number of dimensions. To display the scree plot, press the Scree Plot button. The plot illustrates that the magnitude of the eigenvalues falls off for the first two dimensions; then the plot flattens out for the third and remaining dimensions. From this graph, two dimensions appear appropriate. After closing the Scree Plot window, press the OK button to perform the analysis. See Figure 10.

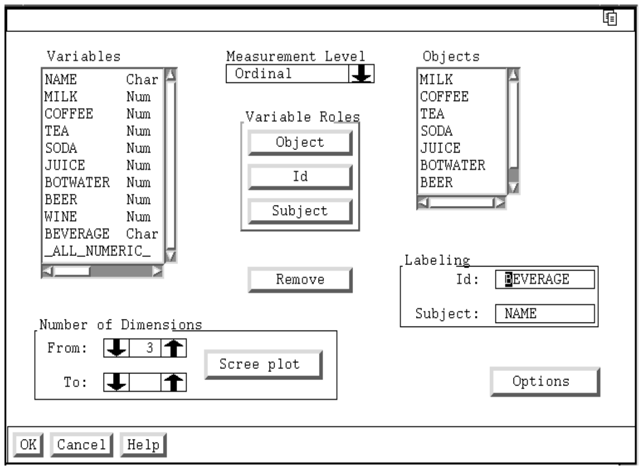


Figure 11. MDS Variable Selection

Results. The plot on the main analysis window contains points for the 17 car models and vectors for the 25 respondents. Interpretations of the two dimensions are 1) the vertical dimension separates foreign and domestic cars in the upper half and lower half, respectively, and 2) the horizontal dimension separates small cars and big cars in the left and right halves, respectively. Respondents prefer cars whose points are closest to their vector. Notice that there are a number of vectors in the upper right quadrant of the plot but there are no cars. This lack of available products to satisfy peoples' preferences indicates a possible niche to fill.

Other results are the "Initial Eigenvalue Plot," "Final Eigenvalue Plot," and "Configuration Table." The Initial Eigenvalue plot is the same as the scree plot on the Variable Selection window. The Final Eigenvalue plot is also a scree plot; it differs from the initial plot only if a measurement type other than Metric is used. The Configuration Table contains the coordinates for the car points.

Multidimensional Scaling

Multidimensional Scaling (MDS) takes subjects' judgments of either similarity or difference of pairs of items and produces a map of the perceived relationship among items.

For example, suppose you ask seven subjects to state their perceived similarity on a 1 to 7 scale for pairs of beverages, with 1 meaning very similar and 7 meaning very different. The beverages are milk, coffee, tea, soda, juice, bottled water, beer, and wine. Someone may state that their perceived similarity between coffee and tea is 3, somewhat similar, or 7, very different. There are 28 possible pairs of these eight beverages.

The data are ordered in an eight observation by eight variable matrix with one matrix (eight observations) for each subject. On the Data Set Selection window, select the BEVERAGE data set in the SASUSER library, then press the OK button. A message window informs you that MDS requires either similarity or distance data. Press the Continue button.

On the Variables Selection window, select the variables MILK, COFFEE, TEA, SODA, JUICE, BOTWATER, BEER, AND WINE as the objects. See Figure 11. It is crucial that the order of the objects is the same as their order in the rows of each matrix. In other words, from the above order, the upper left corner element in the matrix is MILK, MILK (which has a distance of zero) and the element to its

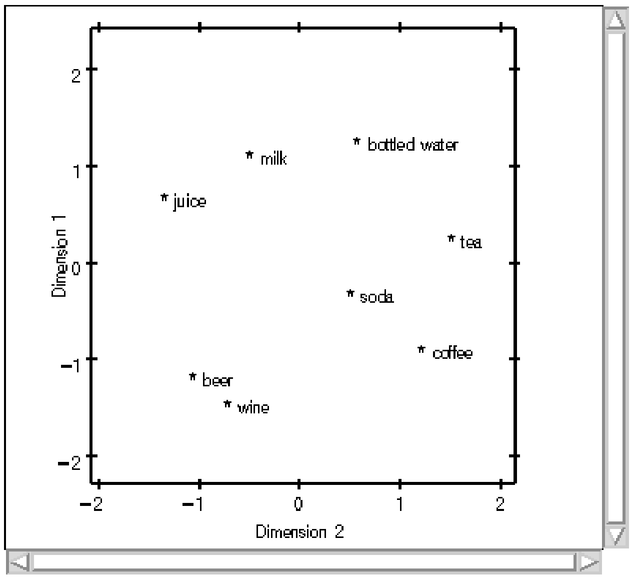


Figure 12. MDS Coordinates Plot

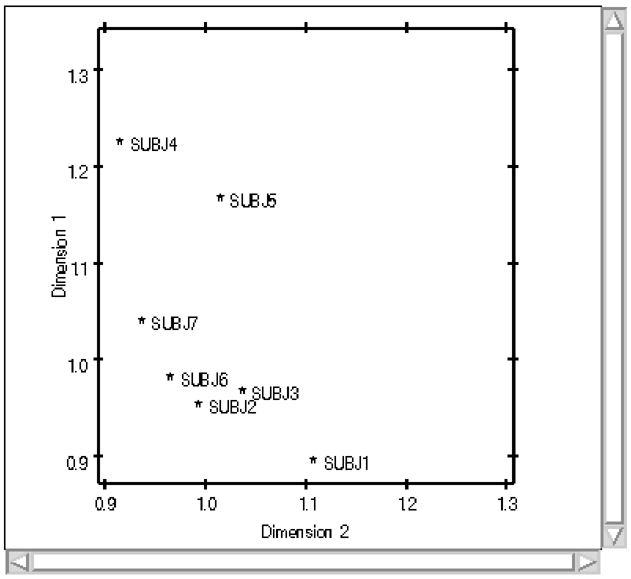


Figure 13. MDS Individual Coefficients Plot

right is MILK, COFFEE.

Also, select BEVERAGE, the beverage names, as the ID variable and NAME, the subject identifiers, as the SUBJECT variable. Because the objects are ordinally-scaled, the ordinal measurement level, the default, is appropriate for this example.

If you think that your subjects may use different perceptual schemes for judging similarity, you can choose to perform an individual differences analysis. Press the Options button and select "Individual Differences Analysis." The data are distances, the default, because larger numbers represent more difference (less similarity). If the data were similarities, you would choose the appropriate selection on the Options window. To close the options window, press the OK button.

As in correspondence analysis and MDPREF analysis, you can set the number of dimensions for the solution. With MDS you have an extra capability; you can solve for several dimensional solutions in one analysis.

Choose a three-dimensional solution by entering a "3" in the input field to the right of the "From:" label or by clicking on the up arrow to its right until the number 3 appears in the input field. As with the other dimensional analyses, a scree plot may be useful in determining the appropriate number of dimensions. You can create the plot by pressing the Scree Plot button.

To continue with the analysis, press the OK button on the Variable Selection window.

Results. As with the correspondence analysis and MDPREF plots, interpreting the MDS plot has two parts: 1) finding a reasonable interpretation for each of the plot dimensions, and 2) finding a reasonable interpretation of the relationship of the points in the plot. See Figure 12.

The presence of bottled water, milk, and juice at the top of the plot and wine, beer, and coffee at the bottom of the plot might indicate a good for you/not so good for you interpretation for the vertical dimension, Dimension 1. The horizontal dimension, Dimension 2, does not have as clear an interpretation. Try to come up with your own interpretation that would have tea, coffee, and water on one side and juice, beer, and wine on the other.

Because you requested a three-dimensional solution, two other plots can be displayed: Dimensions 1 and 3 and Dimensions 2 and 3. To change which dimensions are plotted, press the Plot button and select the desired dimensions. Also, on this window you can choose to display the coefficients of the individual differences analysis instead of the coordinates. To do so, select "Coefficient" at the bottom of the window and press the OK button.

In an individual differences analysis, there is a common perceptual map for all subjects, but different subjects have different weights for each dimension. See Figure 13. SUBJ4 is found to be highest on the vertical axis and lowest on the horizontal axis. In other words, SUBJ4 weights whatever this dimension represents more than do the other subjects and it weights whatever dimension 2 represents less than the other subjects. If the good-for-you interpretation is appropriate for Dimension 1, then it plays a larger role in SUBJ4's perceptual mapping of these beverages than it does for other subjects.

It is possible that SUBJ1, SUBJ2, SUBJ3, SUBJ6, and SUBJ7 may cluster together and SUBJ4 and SUBJ5 may be outliers. Additional subjects may sharpen this possible clustering or eliminate it. MDS is useful in market research for discovering possible perceptual perspectives used by consumers and for revealing possible market segments.

You can display other results by pressing the Results button. These results include Fit statistics, Configuration tables, Residual plots, and the Iteration history. The fit statistics are measures of how well the data fit the model. The Configuration tables contain the coordinates and, optionally, the individual difference coefficients that are used in the plots. The Residual plots allow you to assess the fit of the model graphically. The iteration history contains information about how many iterations were needed and how the criterion changed over the iterations.

Summary

Investigators in the field of market research are interested in how consumers make decisions when they choose to buy products. What attributes are important? Do all people make decisions in the same way? If not, how do they differ? What are the perceptual schemes that people use in their purchasing decisions?

The analyses described in this paper can be used with many different types of data to investigate these questions. The Market Research application makes these analyses easy to use, and it is available in Release 6.11 and subsequent releases with the SAS/STAT product.

Acknowledgments

I would like to thank Greg Goodwin, Warren Kuhfeld, Julie LaBarr, Donna Sawyer, and Maura Stokes for their thoughtful comments on this paper.

Experimental Design: Efficiency, Coding, and Choice Designs

Warren F. Kuhfeld

Abstract

This chapter discusses some of the fundamental concepts in marketing research experimental design including factorial designs, orthogonal arrays, balanced incomplete block designs, nonorthogonal designs, and choice and conjoint designs. Design terminology is introduced, design efficiency and the relationship between designs for linear and choice models are explained, and several examples of constructing designs for marketing research choice experiments are presented.[*]

You should familiarize yourself with the concepts in this chapter before studying the conjoint chapter (page 681) or the discrete choice chapter (page 285). After you are comfortable with the material in this chapter, consider looking at the other design chapters starting on pages 243 and 265.

Introduction

An *experimental design* is a plan for running an experiment, and it is often displayed as a matrix. The *factors* of an experimental design are the columns or variables that have two or more fixed values or *levels*. The rows of a design are the treatment combinations and are sometimes called *runs*. Experiments are performed to study the effects of the factor levels on a dependent or response variable.

Experimental designs are important tools in marketing research, conjoint analysis, and choice modeling. In a consumer product study, the rows or runs correspond to product profiles and the factors correspond to the attributes of the hypothetical products or services. In a conjoint study, the rows of the design correspond to products, and the dependent variable or response is a rating or a ranking of the products. In a discrete-choice study, the rows of the design correspond to product *alternatives*. The choice design consists of blocks of several alternatives, and each set of alternatives is called a *choice set*. The dependent variable or response is choice (product i was chosen and the other products in the set were not chosen). See page 681 for an introduction to conjoint analysis and page 289 for an introduction to choice models. The next two sections show simple examples of conjoint and choice experiments.

[*]Copies of this chapter (MR-2010C), the other chapters, sample code, and all of the macros are available on the Web `http://support.sas.com/resources/papers/tnote/tnote_marketresearch.html`. Specifically, sample code is here `http://support.sas.com/techsup/technote/mr2010c.sas`. For help, please contact SAS Technical Support. See page 25 for more information. Parts of this chapter are based on the tutorial that Don Anderson and Warren F. Kuhfeld presented for many years at the American Marketing Association's Advanced Research Techniques Forum.

The Basic Conjoint Experiment

A conjoint study uses experimental design to create a list of products, and subjects rate or rank the products. The conjoint analysis model is a linear model of the form $\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$ where $\mathbf{y}$ contains product ratings or rankings, $\mathbf{X}$ is the coded design matrix (see pages 70 and 73), and $\hat{\boldsymbol{\beta}}$ is the vector of parameter estimates or part-worth utilities. The following table displays a conjoint design and the layout of a simple conjoint experiment with two factors:

Conjoint Design

Acme	$1.99
Acme	$2.99
Ajax	$1.99
Ajax	$2.99
Comet	$1.99
Comet	$2.99

Full-Profile Conjoint Experiment

		Rate Your Purchase Interest
Acme	$1.99	☐
Acme	$2.99	☐
Ajax	$1.99	☐
Ajax	$2.99	☐
Comet	$1.99	☐
Comet	$2.99	☐

In a real experiment, the product descriptions are typically more involved and might use art or pictures, but the basic experiment involves people seeing products and rating or ranking them. The brand attribute has three levels, Acme, Ajax, and Comet, and the price attribute has two levels, $1.99 and $2.99. There are a total of six products.

The Basic Choice Experiment

A discrete choice study uses experimental design to create sets of products, and subjects choose a product from each set. Like the conjoint model, the choice model has a linear utility function, but it is embedded in a nonlinear model (see page 71). The following table displays a choice design and the layout of a simple choice experiment:

Choice Design

1	Acme	$2.99
	Ajax	$1.99
	Comet	$1.99
2	Acme	$2.99
	Ajax	$2.99
	Comet	$2.99
3	Acme	$1.99
	Ajax	$1.99
	Comet	$2.99
4	Acme	$1.99
	Ajax	$2.99
	Comet	$1.99

Discrete Choice Experiment

1	2	3	Choice
Acme $2.99	Ajax $1.99	Comet $1.99	
Acme $2.99	Ajax $2.99	Comet $2.99	
Acme $1.99	Ajax $1.99	Comet $2.99	
Acme $1.99	Ajax $2.99	Comet $1.99	

In a real experiment, the product descriptions are typically more involved and they might use art or pictures, but the basic experiment involves people seeing sets of products and making choices. This example has four choice sets, each composed of three alternative products; so subjects make four choices. Each alternative is composed of two attributes: brand has three levels, and price has two levels.

Attributes can be generic or alternative-specific. A *generic attribute* is treated the same way for each brand, like the price attribute in the design on the left. A design that consists entirely of generic attributes is called a generic design. An *alternative-specific attribute* is analyzed separately for each brand, such as the set of price factors in the design on the right. Note that the alternative-specific price effects consist of the interaction (product) of the binary brand effects and the generic price effect.

Coded Choice Design With a Generic Price Effect

1	Acme	1	0	0	$2.99
	Ajax	0	1	0	$1.99
	Comet	0	0	1	$1.99
2	Acme	1	0	0	$2.99
	Ajax	0	1	0	$2.99
	Comet	0	0	1	$2.99
3	Acme	1	0	0	$1.99
	Ajax	0	1	0	$1.99
	Comet	0	0	1	$2.99
4	Acme	1	0	0	$1.99
	Ajax	0	1	0	$2.99
	Comet	0	0	1	$1.99

Coded Choice Design With Alternative-Specific Price Effects

1	Acme	1	0	0	$2.99	0	0
	Ajax	0	1	0	0	$1.99	0
	Comet	0	0	1	0	0	$1.99
2	Acme	1	0	0	$2.99	0	0
	Ajax	0	1	0	0	$2.99	0
	Comet	0	0	1	0	0	$2.99
3	Acme	1	0	0	$1.99	0	0
	Ajax	0	1	0	0	$1.99	0
	Comet	0	0	1	0	0	$2.99
4	Acme	1	0	0	$1.99	0	0
	Ajax	0	1	0	0	$2.99	0
	Comet	0	0	1	0	0	$1.99

Chapter Overview

This chapter begins with an introduction to experimental design and choice design. Then it presents examples of the basic approaches to choice design. In all examples, the data are analyzed with a multinomial logit model. The examples include the following:

- The first example creates a design where all of the attributes of all of the alternatives are balanced and orthogonal. This design, which is efficient for a hypothetical linear model involving all of the attributes of all of the alternatives, is converted to the choice design format. This is a useful approach for alternative-specific models and models with cross-effects or when you are not sure what model you will ultimately use in your analysis. See page 127.
- The second example is a continuation of the first example. In this case, a design that is fully balanced and orthogonal cannot be used since there are restrictions on the design. This example uses the same tools that the first example uses, but it additionally uses a macro and options to impose restrictions on the design. See page 156.
- The third example searches a candidate set of alternatives for a design that is efficient for a specific choice model under the null hypothesis $\boldsymbol{\beta} = \mathbf{0}$. This approach is useful for generic models and whenever you are willing to specify a specific model and parameter vector before you collect your data. See page 166.
- The fourth example is a variation on the third example. This examples searches a candidate set of alternatives for a design that is efficient for a specific choice model under the null hypothesis $\boldsymbol{\beta} = \mathbf{0}$. Restrictions are imposed within each choice set across the alternatives. This approach is useful for restricted models such as restricted generic models and whenever you are willing to specify a specific model and parameter vector before you collect your data. See page 177.
- The fifth example searches a candidate set of choice sets for a design that is efficient for a specific choice model under the null hypothesis $\boldsymbol{\beta} = \mathbf{0}$. This approach can be used for generic models and whenever you are willing to specify a specific model and parameter vector before you collect your data. It is sometimes used when there are restrictions within choice sets but across alternatives. For example, this approach can be used when there are restrictions that certain levels in one alternative should not appear with certain levels in another alternative. In contrast, in the third and fourth examples, the design is constructed from a candidate set of alternatives rather than choice sets. Note, however, that the approach illustrated in the fourth example, searching a candidate set of alternatives with restrictions, is often a better approach. See page 188.
- The sixth example creates an efficient design for a purely generic experiment—an experiment that involves no brands, just bundles of attributes. The design is efficient for a choice model under the null hypothesis $\boldsymbol{\beta} = \mathbf{0}$ and for a main-effects model. The design is constructed from a candidate set of alternatives. See page 198. Also see page 102 for information about optimal generic designs. See page 198.
- The seventh example creates an optimal design for a partial-profile choice experiment (Chrzan and Elrod 1995). In each choice set, only a subset of the attributes vary and the rest remain constant. The design is optimal for a partial-profile choice model under the null hypothesis $\boldsymbol{\beta} = \mathbf{0}$ and a main-effects model. The design is constructed from a balanced incomplete block design and an orthogonal array. See page 207.

- The eighth example creates a balanced incomplete block design and uses it in a MaxDiff study (Louviere 1991, Finn and Louviere 1992). Subjects choose their most and least favorite attributes from each set, which is a subset of the full list of attributes. See page 225.

Experimental Design Terminology

The following tables display the conjoint design in four forms:

Full-Factorial Design	
x1	x2
1	1
1	2
2	1
2	2
3	1
3	2

Full-Profile Conjoint Design	
Brand	Price
Acme	1.99
Acme	2.99
Ajax	1.99
Ajax	2.99
Comet	1.99
Comet	2.99

Randomized Design	
x1	x2
2	2
1	1
1	2
3	1
3	2
2	1

Randomized Conjoint Design	
x1	x2
Ajax	2.99
Acme	1.99
Acme	2.99
Comet	1.99
Comet	2.99
Ajax	1.99

The first table contains a “raw” experimental design with two factors. The second contains the same design with factor names and levels assigned. The third contains a randomized version of the raw design. Finally, the fourth contains the randomized design with factor names and levels assigned.

Before an experimental design such as this is used, it should be *randomized.* Randomizing involves sorting the rows into a random order and randomly reassigning all of the factor levels. It is not unusual for the first row of the original design to contain all ones, the first level. Many other groupings or orderings can occur in the original design. Randomization mixes up the levels and eliminates systematic groupings and orderings. For example, randomizing a three level factor changes the original levels (1 2 3) to one of the following: (1 2 3), (1 3 2), (2 1 3), (2 3 1), (3 1 2), (3 2 1). See page 93 for more about randomization, when it is required, and when it is not.

The design in this conjoint example is a *full-factorial design.* It consists of all possible combinations of the levels of the factors. Full-factorial designs let you estimate main effects and interactions. A *main effect* is a simple effect, such as a price or brand effect (see Figure 1). For example, in a main-effects model the brand effect is the same at the different prices and the price effect is the same for the different brands. *Interactions* involve two or more factors, such as a brand by price interaction (see Figure 2). In a model with interactions brand preference is different at the different prices and the price effect is different for the different brands. In Figure 1, there is a main effect for price, and utility increases by one when price goes from \$2.99 to \$1.99 for all brands. Similarly, the change in utility from Acme to Ajax to Comet does not depend on price. In contrast, there are interactions in Figure 2, so the price effect is different depending on brand, and the brand effect is different depending on price.

In a full-factorial design, all main effects, all two-way interactions, and all higher-order interactions are estimable and uncorrelated. The problem with a full-factorial design is that it is too cost-prohibitive and tedious to have subjects consider all possible combinations, for most practical situations. For example, with five factors, two at four levels and three at five levels (denoted $4^2 5^3$), there are $4 \times 4 \times 5 \times 5 \times 5 = 2000$ combinations in the full-factorial design. For this reason, researchers often use *fractional-factorial designs*, which have fewer runs than full-factorial designs. The price of having fewer runs is that some

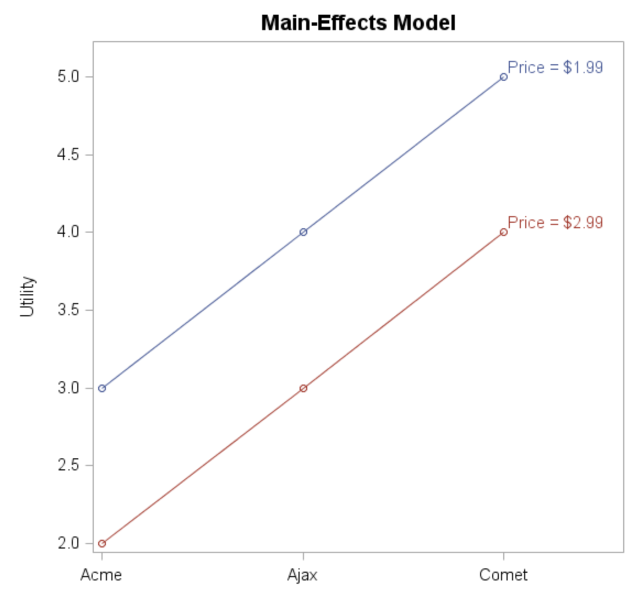


Figure 1

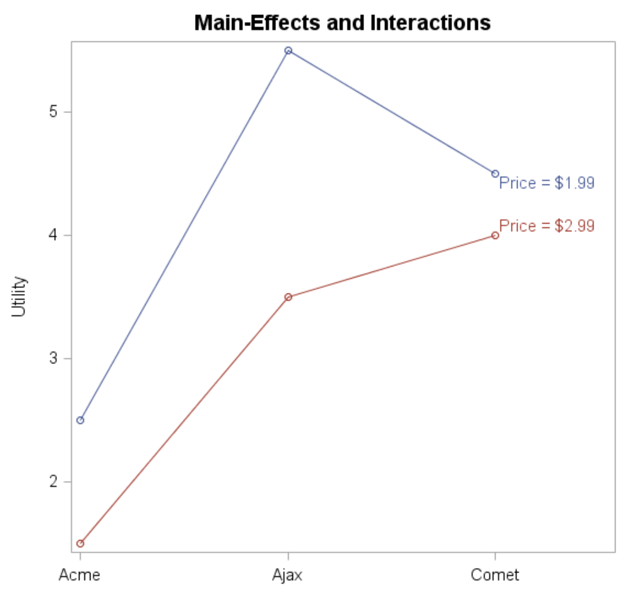


Figure 2

effects become confounded. Two effects are *confounded* or *aliased* when they are not distinguishable from each other. This means that lower-order effects such as main effects or two-way interactions might be aliased with higher-order interactions in most of our designs. We estimate lower-order effects by assuming that higher-order effects are zero or negligible. See page 495 for an example of aliasing.

Fractional-factorial designs that are both orthogonal and balanced are of particular interest. A design is *balanced* when each level occurs equally often within each factor, which means that the intercept is orthogonal to each effect. When every *pair* of levels occurs equally often across all pairs of factors, the design is *orthogonal*. More generally, a design is orthogonal when the frequencies for level pairs are proportional or equal. For example, with 2 two-level factors, an orthogonal design could have pairwise frequencies proportional to 2, 4, 4, 8. Such a design is not balanced—one level occurs twice as often as the other. Imbalance is a generalized form of nonorthogonality; hence it increases the variances of the parameter estimates and decreases the efficiency or goodness of the design.

Fractional-factorial designs are categorized by their *resolution*. The resolution identifies which effects (possibly including interactions) are estimable. For example, for resolution III designs, all main effects are estimable free of each other, but some of them are confounded with two-factor interactions. For resolution IV designs, all main effects are estimable free of each other and free of all two-factor interactions, but some two-factor interactions are confounded with other two-factor interactions. For resolution V designs, all main effects and two-factor interactions are estimable free of each other. More generally, if resolution (r) is odd, then effects of order $e = (r - 1)/2$ or less are estimable free of each other. However, at least some of the effects of order e are confounded with interactions of order $e + 1$. If r is even, then effects of order $e = (r - 2)/2$ are estimable free of each other and are also free of interactions of order $e + 1$. Higher resolutions require larger designs. Resolution III fractional-factorial designs are frequently used in marketing research. They are more commonly known as orthogonal arrays.

Orthogonal Arrays

A special type of factorial design is the *orthogonal array.* In an orthogonal array, all estimable effects are uncorrelated. Orthogonal arrays come in specific numbers of runs for specific numbers of factors with specific numbers of levels. The following list contains all (main effects only) orthogonal arrays up to 28 runs:

4	2^3	12	2^{11}	16	2^{15}	18	$2^1 3^7$	21	$3^1 7^1$	24	2^{23}	25	5^6
6	$2^1 3^1$		$2^4 3^1$		$2^{12} 4^1$		$2^1 9^1$	22	$2^1 11^1$		$2^{20} 4^1$	26	$2^1 13^1$
8	2^7		$2^2 6^1$		$2^9 4^2$		$3^6 6^1$				$2^{16} 3^1$	27	3^{13}
	$2^4 4^1$		$3^1 4^1$		$2^8 8^1$	20	2^{19}				$2^{14} 6^1$		$3^9 9^1$
9	3^4	14	$2^1 7^1$		$2^6 4^3$		$2^8 5^1$				$2^{13} 3^1 4^1$	28	2^{27}
10	$2^1 5^1$	15	$3^1 5^1$		$2^3 4^4$		$2^2 10^1$				$2^{12} 12^1$		$2^{12} 7^1$
					4^5		$4^1 5^1$				$2^{11} 4^1 6^1$		$2^2 14^1$
											$3^1 8^1$		$4^1 7^1$

The list shows the number of runs followed by the design. The design is represented as the number of levels raised a power equal to the number of factors. For example, the first design in the list (2^3) has 3 two-level factors in four runs, and the second design ($2^1 3^1$) has 1 two-level factor and 1 three-level factor in 6 runs. The first five designs in the list are as follows:

2^3		
1	1	1
2	1	2
1	2	2
2	2	1

$2^1 3^1$	
1	1
1	2
1	3
2	1
2	2
2	3

2^7						
1	1	1	1	1	1	1
2	1	2	1	2	1	2
1	2	2	1	1	2	2
2	2	1	1	2	2	1
1	1	1	2	2	2	2
2	1	2	2	1	2	1
1	2	2	2	2	1	1
2	2	1	2	1	1	2

$2^4 4^1$				
1	1	1	1	1
1	2	1	2	2
1	1	2	2	3
1	2	2	1	4
2	2	2	2	1
2	1	2	1	2
2	2	1	1	3
2	1	1	2	4

3^4			
1	1	1	1
1	2	3	2
1	3	2	3
2	2	2	1
2	3	1	2
2	1	3	3
3	3	3	1
3	1	2	2
3	2	1	3

Each of these designs is balanced—each level occurs the same number of times within each factor. Each of these designs is orthogonal—every pair of levels occurs the same number of times across all of the pairs of factors in each design. In the first design, each of the four pairs appears once across all three pairs of factors. In the second design, each of the six pairs appears once. In the third design, each of the four pairs appears twice across the 21 pairs of factors. In the fourth design, each of the four pairs appears twice across the six pairs of two-level factors, and each of the eight pairs appears once across the four pairs that involve the four-level factor and each of the two-level factors. In the fifth design, each of the nine pairs appears once across the six pairs of factors. Since orthogonal arrays are both balanced and orthogonal, they are 100% efficient and optimal. Efficiency, which is explained starting on page 62, is a measure of the goodness of the experimental design.

The term "orthogonal array," is sometimes used imprecisely. It is correctly used to refer to designs that are both orthogonal and balanced, and hence optimal. However, the term is sometimes incorrectly used to refer to designs that are orthogonal but not balanced, and hence not 100% efficient and sometimes not even optimal. Such designs are made from orthogonal arrays by "coding down." Coding down consists of replacing an a-level factor by a b-level factor where $a > b$. For example, coding down might replace a three-level factor by a two-level factor (e.g., replacing all 3's with 1's), or a four-level factor by a three-level factor (e.g., replacing all 4's with 1's), or a five-level factor by a three-level factor (e.g., replacing all 4's with 1's and all 5's with 3's), and so on. Coding down introduces imbalance. It would be more precise to call designs such as these something like "unbalanced arrays."

Orthogonal designs are often practical for main-effects models when the number of factors is small and the number of levels of each factor is small. However, there are some situations in which orthogonal designs might not be practical, such as when not all combinations of factor levels are feasible or make sense, the desired number of runs is not available in an orthogonal design, or a model with interactions is being used. When an orthogonal and balanced design is not practical, you must make a choice. One choice is to change the factors and levels to fit some known orthogonal design. This choice is undesirable for obvious reasons. When a suitable orthogonal and balanced design does not exist, efficient nonorthogonal designs can be used instead. Often, these designs are superior to orthogonal but unbalanced designs.

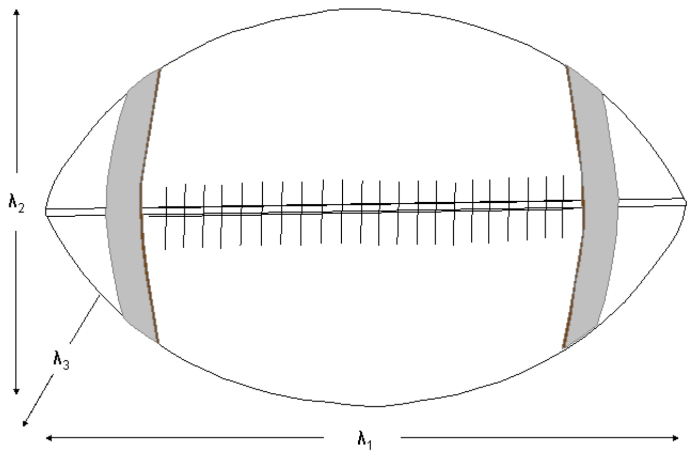
Nonorthogonal designs, where some coefficients might be slightly correlated, can be used when orthogonal designs are not available. You do not have to adapt every experiment to fit some known orthogonal array. First you choose the number of runs. You are not restricted by the sizes of orthogonal arrays, which come in specific numbers of runs for specific numbers of factors with specific numbers of levels. Then you specify the levels of each of the factors and the number of runs. Algorithms for generating efficient designs select a set of *design points* from a set of *candidate points* (such as a full-factorial design or a fractional-factorial design). Design points are selected or replaced when that operation increases an efficiency criterion.*

Throughout this book, we use the `%MktEx` macro to find good, efficient experimental designs. It has specialized algorithms for finding both orthogonal and nonorthogonal designs. The `%MktEx` macro is a part of the SAS autocall library. See page 803 for information about installing and using SAS autocall macros. There are many different construction methods for creating orthogonal arrays. In practice, since we always use the `%MktEx` macro to make them, we never have to worry about the different methods. However, if you would like to learn more, you can read the section beginning on page 95.

Eigenvalues, Means, and Footballs

The next section discusses experimental design efficiency. To fully understand that section, you need some basic understanding of *eigenvalues* and various types of means or averages. This section explains these and other concepts, but without a high degree of mathematical rigor. An American football provides a nice visual image for understanding the eigenvalues of a matrix.

*In the coordinate exchange algorithm that the `%MktEx` macro uses, the full-factorial candidate set is virtual. It is never created, but all combinations in it are available for consideration. In contrast, in the Modified Fedorov search that PROC OPTEX does (either by itself or through the `%MktEx` macro), an explicit full-factorial or fractional-factorial candidate set is created and searched.



The rows or columns of a matrix can be thought of as a swarm of points in Euclidean space. Similarly, a football consists of a set of points in a space. Sometimes, it is helpful to get an idea of the size of your group of points. For a football, you might think of three measures because a football is a three-dimensional object: the longest length from end to end, the height in the center and perpendicular to the length, and finally the width, which for a fully-inflated football is the same as the height. One can do similar things for matrices, and that is where eigenvalues come in. For many of us, eigenvalues are most familiar from factor analysis and principal component analysis. In principal component analysis, one rotates a cloud of points to a principal axes orientation, just as this football has been rotated so that its longest dimension is horizontally displayed. The principal components correspond to: the longest squared length, the second longest squared length perpendicular or orthogonal to the first, the third longest squared length orthogonal to the first two, and so on. The eigenvalues are the variances of the principal components and are proportional to squared lengths. The eigenvalues provide a set of measures of the size of a matrix, just as the lengths provide a set of measures of the size of a football.

The following matrices show a small experimental design, the coded design **X**, the sum of squares and cross products matrix $\mathbf{X'X}$, the matrix inverse $(\mathbf{X'X})^{-1}$, and the eigenvalues of the inverse, Λ:

Design		
1	1	1
1	2	2
1	2	2
2	1	2
2	2	1
2	2	1

X			
1	1	1	1
1	1	−1	−1
1	1	−1	−1
1	−1	1	−1
1	−1	−1	1
1	−1	−1	1

X′X			
6	0	−2	0
0	6	0	−2
−2	0	6	0
0	−2	0	6

$(\mathbf{X'X})^{-1}$			
0.188	0.000	0.063	0.000
0.000	0.188	0.000	0.063
0.063	0.000	0.188	0.000
0.000	0.063	0.000	0.188

Λ			
1/4	0	0	0
0	1/4	0	0
0	0	1/8	0
0	0	0	1/8

X is made from the raw design by coding, which in this case simply involves creating an intercept, appending the design, and replacing 2 with –1. See page 73 for more about coding. The $\mathbf{X'X}$ matrix comes from a matrix multiplication of the transpose of **X** times **X**. For example, the –2 in the first row comes from $\mathbf{x}_1'\mathbf{x}_3 = (1\ 1\ 1\ 1\ 1\ 1)'(1\ -1\ -1\ \ 1\ -1\ -1) = 1\times1+1\times-1+1\times-1+1\times1+1\times-1+1\times-1 = -2$. Explaining the computations involved in finding the matrix inverse and eigenvalues is beyond the scope of this chapter; however, they are explained in many linear algebra and multivariate statistics texts.

The trace is the sum of the diagonal elements of a matrix, which for $(\mathbf{X}'\mathbf{X})^{-1}$, is both the sum of the variances and the sum of the eigenvalues: $\text{trace}\,((\mathbf{X}'\mathbf{X})^{-1}) = \text{trace}\,(\Lambda) = 0.188+0.188+0.188+0.188 = 1/4 + 1/4 + 1/8 + 1/8 = 0.75$. The determinant of $(\mathbf{X}'\mathbf{X})^{-1}$, denoted $|(\mathbf{X}'\mathbf{X})^{-1}|$, is the product of the eigenvalues and is 0.0009766: $|(\mathbf{X}'\mathbf{X})^{-1}| = |\Lambda| = 1/4 \times 1/4 \times 1/8 \times 1/8 = 0.0009766$. The determinant of a matrix is geometrically interpreted in terms of the volume of the space defined by the matrix. The formula for the determinant of a nondiagonal matrix is complicated, so when the eigenvalues are known, determinants are more conveniently expressed as a function of the eigenvalues.

Given a set of eigenvalues, or any set of numbers, we frequently want to create a single number that summarizes the values in the set. The most obvious way to do this is to compute the average or arithmetic mean. The familiar *arithmetic mean* is found by adding together p numbers and then dividing by p. A trace, divided by p, is an arithmetic mean. The arithmetic mean is an enormously popular and useful statistic, however it is not the only way to average numbers. The less familiar *geometric mean* is found by multiplying p numbers together and then taking the pth root of the product. The pth root of a determinant is a geometric mean of eigenvalues. To better understand the geometric mean, consider an example. Say your investments increased by 7%, 5%, and 12% over a three year period. The arithmetic mean of these numbers, $(7 + 5 + 12)/3 = 8\%$, is *not* the average increase that would have come if the investments had increased by the same amount every year. To find that average, we need the geometric mean of the ratios of the current to previous values: $(1.07 \times 1.05 \times 1.12)^{1/3} = 1.0796$. The average increase is 7.96%.

Experimental Design Efficiency

This section discusses precisely what is meant by an efficient design. While this section is important, it is not critical that you understand every mathematical detail. The concepts are explained again in a more intuitive and less mathematical way in the next section. Also, see page 243 for more information about efficient experimental designs.

The goodness or *efficiency* of an experimental design can be quantified. Common measures of the efficiency of an $(N_D \times p)$ design matrix $\mathbf{X}$ are based on the *information matrix* $\mathbf{X}'\mathbf{X}$. The variance-covariance matrix of the vector of parameter estimates $\hat{\boldsymbol{\beta}}$ in a least-squares analysis is proportional to $(\mathbf{X}'\mathbf{X})^{-1}$. More precisely, it equals $\sigma^2(\mathbf{X}'\mathbf{X})^{-1}$. The variance parameter, σ^2, is an unknown constant. Since σ^2 is constant, it can be ignored (or assumed to equal one) in the discussion that follows. The diagonal elements of $(\mathbf{X}'\mathbf{X})^{-1}$ are the parameter estimate variances, and the standard errors are the square roots of the variances. Since they depend only on $\mathbf{X}$ (and σ^2), they can be reported by design software before any data are collected. An efficient design has a "small" variance matrix, and the eigenvalues of $(\mathbf{X}'\mathbf{X})^{-1}$ provide measures of its "size." The process of minimizing the eigenvalues or variances only depends on the selection of the entries in $\mathbf{X}$ not on the unknown σ^2 parameter.

The two most prominent efficiency measures are based on quantifying the idea of matrix size by averaging (in some sense) the eigenvalues or variances. *A-efficiency* is a function of the arithmetic mean of the eigenvalues, which is also the arithmetic mean of the variances, and is given by $\text{trace}\,((\mathbf{X}'\mathbf{X})^{-1})/p$. A-efficiency is perhaps the most obvious measure of efficiency. As the variances get smaller and the arithmetic mean of the variances of the parameter estimates goes down, A-efficiency goes up. However, as we learned in the previous section, there are other averages to consider. *D-efficiency* is a function of the geometric mean of the eigenvalues, which is given by $|(\mathbf{X}'\mathbf{X})^{-1}|^{1/p}$. Both D-efficiency and A-efficiency are based on the idea of average variance, but in different senses of the word "average." We usually use D-efficiency for two reasons. It is the easier and faster of the two for a computer program to optimize. Furthermore, relative D-efficiency, the ratio of two D-efficiencies for two competing designs, is invariant under different coding schemes. This is not true with A-efficiency. A third common efficiency measure, *G-efficiency*, is based on σ_M, the maximum standard error for prediction over the candidate set. All three of these criteria are convex functions of the eigenvalues of $(\mathbf{X}'\mathbf{X})^{-1}$ and hence are usually highly correlated.

For all three criteria, if a balanced and orthogonal design exists, then it has optimum efficiency; conversely, the more efficient a design is, the more it tends toward balance and orthogonality. A design is balanced and orthogonal when $(\mathbf{X}'\mathbf{X})^{-1}$ is diagonal and equals $\frac{1}{N_D}\mathbf{I}$ for a suitably coded $\mathbf{X}$. A design is orthogonal when the submatrix of $(\mathbf{X}'\mathbf{X})^{-1}$, excluding the row and column for the intercept, is diagonal; there might be off-diagonal nonzeros for the intercept. A design is balanced when all off-diagonal elements in the intercept row and column are zero. How we choose $\mathbf{X}$ determines the efficiency of our design. Ideally, we want to choose $\mathbf{X}$ so that the design is balanced and orthogonal or at least very nearly so. More precisely, we want to choose $\mathbf{X}$ so that we maximize efficiency.

These measures of efficiency can be scaled to range from 0 to 100 (see pages 73–73 for the orthogonal coding of $\mathbf{X}$ that must be used with these formulas) as follows:

$$\begin{aligned}
A\text{-efficiency} &= 100 \times \frac{1}{N_D \ \text{trace}\,((\mathbf{X}'\mathbf{X})^{-1})/p} \\
D\text{-efficiency} &= 100 \times \frac{1}{N_D \ |(\mathbf{X}'\mathbf{X})^{-1}|^{1/p}} \\
G\text{-efficiency} &= 100 \times \frac{\sqrt{p/N_D}}{\sigma_M}
\end{aligned}$$

These efficiencies measure the goodness of a design relative to hypothetical orthogonal designs that might not exist, so they are not useful as absolute measures of design efficiency. Instead, they should be used relatively, to compare one design to another for the same situation. Efficiencies that are not near 100 might be perfectly satisfactory. When D-efficiency is 0, one or more parameters cannot be estimated. When D-efficiency is 100, then the design is balanced and orthogonal. Values in between mean that all of the parameters can be estimated, but with less than optimal precision. Precisely what this means can vary from design to design. It might be that all of the variances are larger than the optimal value, or it might be that only some are larger than the optimal value. When the standardized orthogonal contrast coding on pages 73–73 is used, D-efficiency computed this way can never vary outside the 0 to 100 range. The range for D-efficiency can be quite different (either larger or smaller) with other coding schemes.

Experimental Design: Rafts, Rulers, Alligators, and Stones

A good physical metaphor for understanding experimental design and design efficiency is a raft. A raft is a flat boat, often supported by flotation devices attached to the corners. The raft in Figure 3 has four Styrofoam blocks under each corner, which provide nice stability and equal support. This raft corresponds to 2 two-level factors from a 16-run design (see Table 1). The four corners correspond to each of the four possible combinations of 2 two-level factors, and the four blocks under the raft form an up-side-down bar chart showing the frequencies for each of the four combinations. Looking at the

Table 1
Two-Level Factors in 16 Runs

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	2	2	2	2	1	1	2	2	1	2	2
1	1	2	2	1	1	2	2	1	2	1	2	2	1	2
1	1	2	2	2	2	1	1	1	2	2	1	2	2	1
1	2	1	2	1	2	1	2	2	2	2	2	1	1	1
1	2	1	2	2	1	2	1	2	2	1	1	1	2	2
1	2	2	1	1	2	2	1	2	1	2	1	2	1	2
1	2	2	1	2	1	1	2	2	1	1	2	2	2	1
2	1	1	2	1	2	2	1	2	1	1	2	2	2	1
2	1	1	2	2	1	1	2	2	1	2	1	2	1	2
2	1	2	1	1	2	1	2	2	2	1	1	1	2	2
2	1	2	1	2	1	2	1	2	2	2	2	1	1	1
2	2	1	1	1	1	2	2	1	2	2	1	2	2	1
2	2	1	1	2	2	1	1	1	2	1	2	2	1	2
2	2	2	2	1	1	1	1	1	1	2	2	1	2	2
2	2	2	2	2	2	2	2	1	1	1	1	1	1	1

Table 2
Unbalanced
$2^2 3^3$ in 18 Runs

1	1	1	1	1
1	1	2	3	3
1	1	1	3	2
1	1	3	2	3
1	2	2	2	1
1	2	3	1	2
1	1	1	2	3
1	1	3	3	2
1	1	2	2	2
1	1	3	1	1
1	2	1	3	1
1	2	2	1	3
2	1	2	1	2
2	1	3	2	1
2	1	1	1	3
2	1	2	3	1
2	2	1	2	2
2	2	3	3	3

Table 3

Making Twos from Threes

1		1	1		1
1		1	1		1
1		1	2		2
1		1	2		2
1		1	3		1
1		1	3		1
2		2	1		1
2		2	1		1
2	$\rightarrow$	2	2	$\rightarrow$	2
2		2	2		2
2		2	3		1
2		2	3		1
3		1	1		1
3		1	1		1
3		1	2		2
3		1	2		2
3		1	3		1
3		1	3		1

Table 4
Optimal
$2^2 3^3$ in 18 Runs

1	1	1	2	2
1	1	2	3	2
1	1	3	1	2
1	1	3	2	3
1	2	1	1	3
1	2	1	3	1
1	2	2	1	3
1	2	2	2	1
1	2	3	3	1
2	1	1	2	1
2	1	1	3	3
2	1	2	1	1
2	1	2	3	3
2	1	3	1	1
2	2	1	1	2
2	2	2	2	2
2	2	3	2	3
2	2	3	3	2

raft, one can tell that the first factor is balanced (equal support on the left and on the right) as is the second (equal support in the front and the back). The design is also orthogonal (equal support in all four corners). Making a design that supports your research conclusions is like making a raft for use in water with alligators. You want good support no matter on which portion of the raft you find yourself. Similarly, you want good support for your research and good information about all of your product attributes and attribute levels.

Now compare the raft in Figure 3 to the one shown in Figure 4. The Figure 4 raft corresponds to the two-level factors in the design shown in Table 2. This design has 18 runs, and since 18 cannot be divided by 2×2, a design that is both balanced and orthogonal is not possible. Clearly, this design is not balanced in either factor. There are twelve blocks on the left and only six on the right, and there are twelve blocks on the back and only six on the front. This design *is* however orthogonal, because the corner frequencies are proportional. These two factors can be made from 2 three-level factors in the L_{18} design, which has up to 7 three-level factors. See Table 3. The three-level factors are all orthogonal, and recoding levels, replacing 3 with 1, preserves that orthogonality at the cost of decreased efficiency and a horrendous lack of balance. See Table 3 on page 250 for the information and variance matrices for the Figure 4 raft.

Finally, compare the raft in Figure 4 to the one shown in Figure 5. Both of these correspond to designs with two-level factors in 18 runs. The Figure 5 raft corresponds to a design that is balanced. There are nine blocks on the left and nine on the right, and there are nine blocks on the back and nine on the front. The design is not however orthogonal since the corner frequencies are 4, 5, 4, and 5, which are not equal or even proportional. Ideally, you would like a raft such as the one in Figure 3, which corresponds to a design that is both orthogonal and balanced. However, to have both two or more three-level and two or more two-level factors, you need at least 36 runs. More precisely, you need a multiple of 36 runs (36, 72, 108, and so on). In 18 runs, you can make an optimal design (that is, optimal relative to all other designs in 18 runs), such as the one in Table 4 and Figure 5, that provides good support under all corners but not perfectly equal support. See Tables 3 and 4 in the next chapter on pages 251 and 250 for the information and variance matrices for the Figure 4 and 5 rafts.

On which raft would you rather walk? The Figure 3 and Figure 5 rafts are going to be reasonably stable. The Figure 3 raft is in fact optimal, given exactly 16 Styrofoam blocks, and the Figure 5 raft is also optimal, given exactly 18 Styrofoam blocks. The Figure 4 raft might be fine if you stay in the back left corner, but take one step, and you have little support. An experimental design provides support for your research just as a raft provides support for a person crossing a river. In both raft and design terms, the problem is one of stability and support. In design terms, part of your results are not stable due to a lack of information about the front right combination in your factorial design. How confident can you be in your results when you have so little information about some of your product attribute levels?

The Table 4 design (Figure 5 raft) brings to mind the story of the cup containing exactly one half cup of water. The optimist sees the cup as half full, and the pessimist sees it as half empty. In the design, the optimist sees a little extra support in the back left and front right corners. The pessimist sees a little less support in the front left and back right corners. Either way, all available resources (design points) are optimally allocated to maximize efficiency and stability. What you would really like is both balance and orthogonality. However, you cannot get both in 18 runs, because 2×2 does not divide 18. Still, you can do pretty well. Like the line in the song, “You can’t always get what you want, but if you try sometimes, you just might find, you get what you need” (Jagger and Richards 1969). What you want is orthogonality and balance. What you need is good stability. Efficient designs can give you what you need even when what you want is impossible.

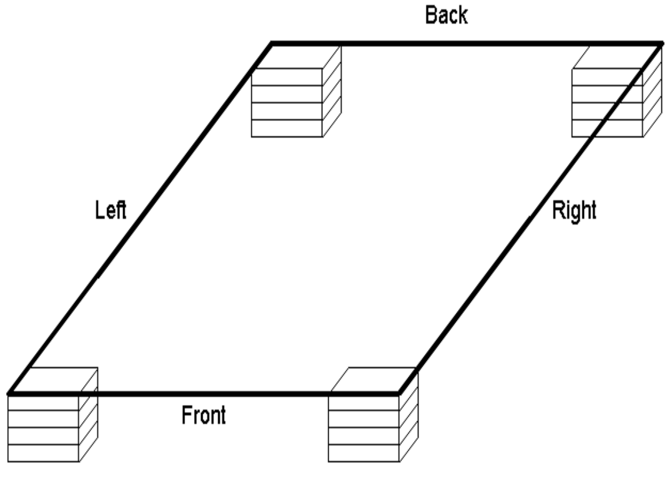


Figure 3 *16 Runs, Orthogonal and Balanced*

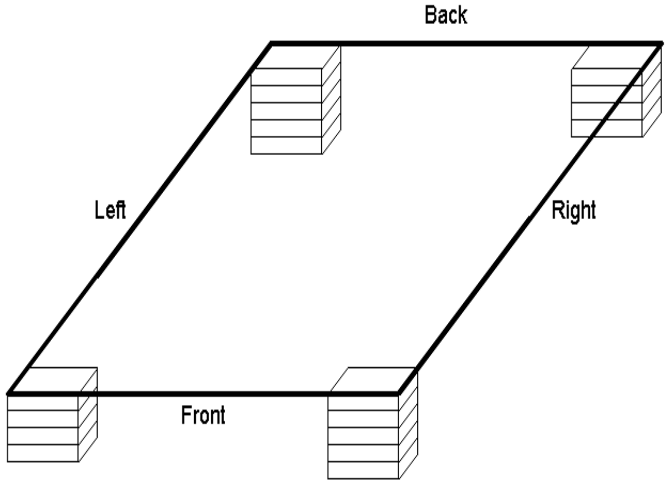


Figure 5 *18 Runs, Balanced and Almost Orthogonal*

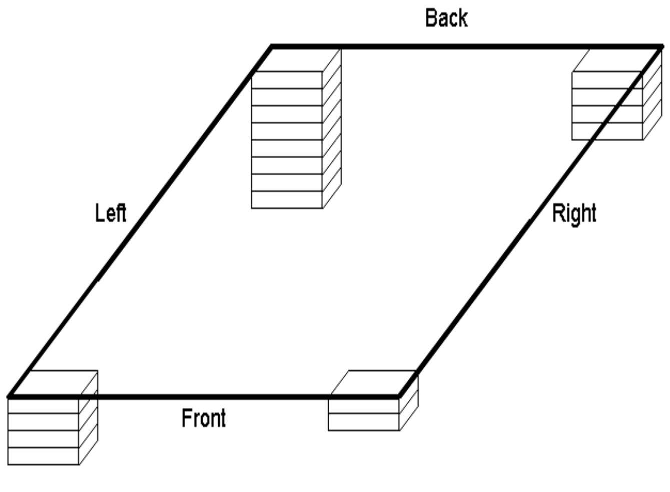
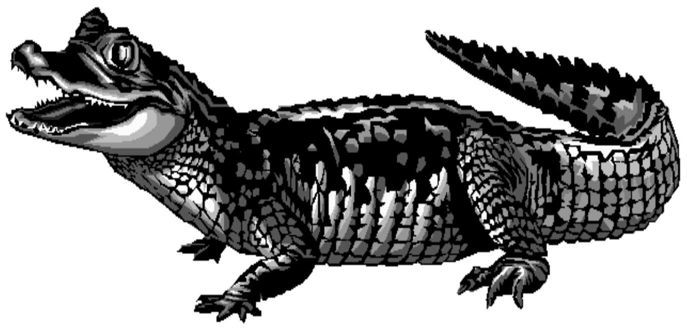


Figure 4 *18 Runs, Orthogonal but not Balanced*



The Table 2 and Figure 4 design might seem like just a "straw man," something that we build up just so that we can knock it down. However, this design was widely used in the past, and in fact, in spite of the fact that its deficiencies have been known for over 16 years (Kuhfeld, Tobias, and Garratt 1994), it is *still* used in some sources as a text-book example of a good design. In fact, it is a text-book example of how *not* to make designs. It is an example of what can happen when you choose orthogonality as the be-all-end-all design criterion, ignoring both balance and statistical efficiency. It is also an example of what can happen when you construct designs from a small, inferior, and incomplete catalog instead of using a comprehensive designer. Even among orthogonal designs, it is not optimal (see pages 249–251). If we achieve perfect orthogonality *and* balance, our design is optimal and has maximum efficiency. The key consideration is that maximizing statistical efficiency minimizes the variability of our parameter estimates, and that is what we want to achieve. Recall that for a linear model, the variance-covariance matrix of the vector of parameter estimates is proportional to $(\mathbf{X}'\mathbf{X})^{-1}$. Maximizing efficiency minimizes those variances, covariances, and hence standard errors. These designs are discussed in more detail, including an examination of their variance matrices, starting on page 249.

How we choose our design, our **X** values, affects the variability of our parameter estimates. Previously, we talked about eigenvalues and the variance matrix, which provides a mathematical representation of the idea that we choose our **X** values so that our parameter estimates have small standard errors. Now, we will discuss this less mathematically. Imagine that we are going to construct a very simple experiment. We are interested in investigating the purchase interest of a product as a function of its price. We design an experiment with two prices, \$1.49 and \$1.50 and ask people to rate how interested they are in the products at those two prices. We plot the results with price on the horizontal axis and purchase interest on the vertical axis. We find that the price effect is minimal. See Figure 6. Now imagine that the line is a ruler and the two dots are your fingers. Your fingers are the design points providing support for your research. Your fingers are close together because in our research design, we chose two prices that are close together. Furthermore, imagine that there is a small amount of error in your data, that is error in the reported purchase interest, which is in the vertical direction. To envision this, move your fingers up and down, just a little bit. What happens to your slope and intercept as you do this?[*] They vary a lot! This is not a function of your data; it is a function of your design being inefficient because you did not adequately sample a reasonable price range.

Next, let's design a similar experiment, but this time with prices of \$0.99 and \$1.99. See Figure 7. Imagine again that the line is a ruler and the two dots are your fingers, but this time they are farther apart. Again, move your fingers up and down, just a little bit. What happens to your slope and intercept as you do this? Not very much; they change a little bit. The standard errors for Figure 6 are much greater than the standard errors for Figure 7. **How you choose your design points affects the stability of your parameter estimates. This is the same lesson that the mathematics involving $(\mathbf{X}'\mathbf{X})^{-1}$ gives you. You want to choose your X's so that efficiency is maximized and the variability of your parameter estimates is minimized.** This example does not imply, however, that you should pick prices such as \$0.01 and \$1,000,000,000. Your design levels need to make sense for the product.

Conjoint, Linear Model, and Choice Designs

Consider a simple example of three brands each at two prices. We always use linear-model theory to guide us in creating designs for a full-profile conjoint studies. Usually we pick orthogonal arrays for

[*]I encourage you to actually try this and see what happens! It is a great physical demonstration showing that you you choose **X** affects the stability of the parameter estimates.

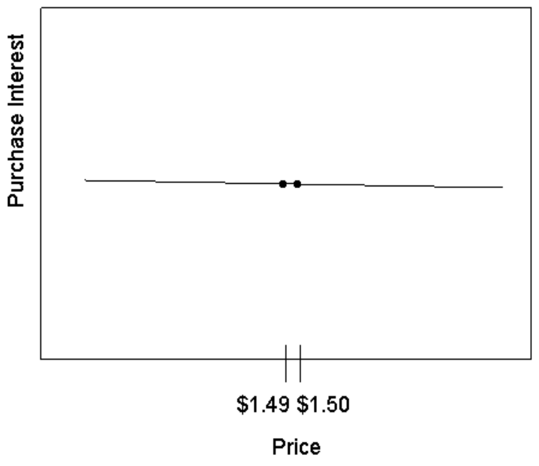


Figure 6 Prices Close Together

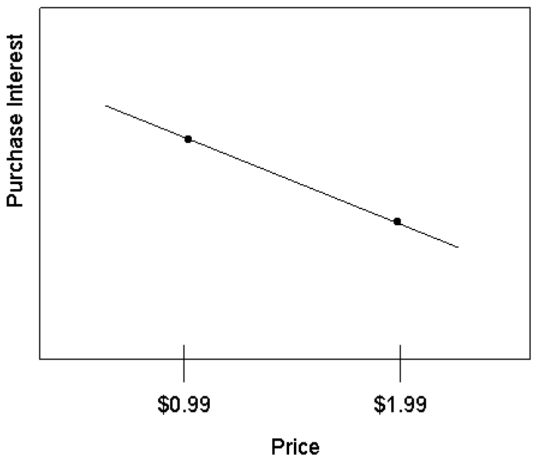


Figure 7 Prices Farther Apart

conjoint studies. For choice modeling, the process is somewhat different. We sometimes use linear-model theory to create a "linear arrangement of a choice design" (or *linear arrangement* for short) from which we then construct a true *choice design* to use in a discrete choice study. The linear arrangement does not correspond to a real and useful linear model. Rather the linear arrangement of a choice design simply provides a convenient way to generate and evaluate choice designs for certain problems such as problems with alternative-specific effects. An example of a conjoint and a linear arrangement of a choice design are as follows:

Full-Profile
Conjoint Design

Brand	Price
1	1.99
1	2.99
2	1.99
2	2.99
3	1.99
3	2.99

Linear Arrangement
of a Choice Design

Brand 1 Price	Brand 2 Price	Brand3 Price
1.99	1.99	1.99
1.99	2.99	2.99
2.99	1.99	2.99
2.99	2.99	1.99

This conjoint design has two factors, brand and price, and six runs or product profiles. Subjects are shown each combination, such as brand 1 at $1.99 and are asked to report purchase interest through either a rating (for example, on a 1 to 9 scale) or a ranking of the six profiles.

The linear arrangement of the choice design for a pricing study with three brands has three factors (Brand 1 Price, Brand 2 Price, and Brand 3 Price) and one row for each choice set. More generally, the linear arrangement has one factor for each attribute of each alternative (or brand), and brand is not a factor in the linear arrangement. Each brand is a "bin" into which its factors are collected. Subjects see these sets of products and report which one they would choose (and implicitly, which ones they

Table 5

Linear Arrangement

1	2	3
1.99	1.99	1.99
1.99	2.99	2.99
2.99	1.99	2.99
2.99	2.99	1.99

Table 6

Choice Design

Set	Brand	Price
1	1	1.99
	2	1.99
	3	1.99
2	1	1.99
	2	2.99
	3	2.99
3	1	2.99
	2	1.99
	3	2.99
4	1	2.99
	2	2.99
	3	1.99

Table 7

Choice Design Coding

	Brand Effects			Brand by Price		
Set	1	2	3	1	2	3
1	1	0	0	1.99	0	0
	0	1	0	0	1.99	0
	0	0	1	0	0	1.99
2	1	0	0	1.99	0	0
	0	1	0	0	2.99	0
	0	0	1	0	0	2.99
3	1	0	0	2.99	0	0
	0	1	0	0	1.99	0
	0	0	1	0	0	2.99
4	1	0	0	2.99	0	0
	0	1	0	0	2.99	0
	0	0	1	0	0	1.99

would not choose). However, before we fit the choice model, we need to construct a true choice design from the linear arrangement and code the choice design. See Tables 5, 6, and 7.

The linear arrangement has one row per choice set. The choice design has three rows for each choice set, one for each alternative. The linear arrangement and the choice design contain different arrangements of the exact same information. In the linear arrangement, brand is a bin into which its factors are collected (in this case one factor per brand). In the choice design, brand and price are both factors, because the design has been rearranged from one row per choice set to one row per alternative per choice set. For this problem, with only one attribute per brand, the first row of the choice design matrix corresponds to the first value in the linear arrangement, Brand 1 at $1.99. The second row of the choice design matrix corresponds to the second value in the linear arrangement, Brand 2 at $1.99. The third row of the choice design matrix corresponds to the third value in the linear arrangement, Brand 3 at $1.99, and so on.

A design is coded by replacing each factor with one more columns of *indicator variables* (which are often referred to as “dummy variables”) or other codings. In this example, a brand factor is replaced by the three binary variables. We go through how to construct and code linear and choice designs many times in the examples using a number of different codings. For now, just notice that the conjoint design is different from the linear arrangement of the choice design, which is different from the customary arrangement of the choice design. They are not even the same size! You can use the `%MktEx` macro to make the linear arrangement, the `%MktRoll` macro to convert it into a choice design, the `%ChoicEff` macro to evaluate the choice design, and the TRANSREG procedure to code the choice design for analysis. Alternatively, you can use the `%ChoicEff` macro to construct and evaluate the choice design directly and never go through the linear arrangement phase. Both approaches are discussed in detail in this chapter and this book.

Table 8
The First Six Choice Sets (Out of 36 Total)

Linear Arrangement of a Choice Design									Choice Design			
Brand 1			Brand 2			Brand 3						
x1	x2	x3	x4	x5	x6	x7	x8	x9		x1	x2	x3
16 oz.	0.89	Twist Up	24 oz.	0.89	Twist Off	20 oz.	0.99	Pop Up	1	16 oz.	0.89	Twist Up
									2	24 oz.	0.89	Twist Off
									3	20 oz.	0.99	Pop Up
20 oz.	0.99	Pop Up	24 oz.	0.89	Twist Up	20 oz.	0.89	Twist Off	1	20 oz.	0.99	Pop Up
									2	24 oz.	0.89	Twist Up
									3	20 oz.	0.89	Twist Off
20 oz.	0.89	Twist Up	20 oz.	0.99	Twist Off	16 oz.	0.89	Twist Off	1	20 oz.	0.89	Twist Up
									2	20 oz.	0.99	Twist Off
									3	16 oz.	0.89	Twist Off
20 oz.	0.89	Twist Up	16 oz.	0.99	Twist Up	24 oz.	0.99	Pop Up	1	20 oz.	0.89	Twist Up
									2	16 oz.	0.99	Twist Up
									3	24 oz.	0.99	Pop Up
16 oz.	0.89	Twist Off	24 oz.	0.99	Pop Up	16 oz.	0.99	Twist Off	1	16 oz.	0.89	Twist Off
									2	24 oz.	0.99	Pop Up
									3	16 oz.	0.99	Twist Off
24 oz.	0.99	Twist Off	16 oz.	0.89	Twist Off	16 oz.	0.89	Pop Up	1	24 oz.	0.99	Twist Off
									2	16 oz.	0.89	Twist Off
									3	16 oz.	0.89	Pop Up

The effects that are labeled in Table 7 as "Brand by Price" are called *alternative-specific effects.* They are coded so that the price effect can be different for each alternative or brand.

A slightly more involved illustration of the differences between the linear and final version of a choice design is shown in Table 8. This example has three brands and three alternatives, one per brand. The product is sports beverages, and they are available in three sizes, at two prices with three different types of tops including a top that pops straight up for drinking, an ordinary twist off top, and cap that twists up for drinking without coming off.

The linear arrangement has one row per choice set. The full choice design has 36 choice sets. There is one factor for each attribute of each alternative. This experiment has three alternatives, one for each of three brands, and three attributes per alternative. The first goal is to make a linear arrangement of a choice design where each attribute, both within and between alternatives, is orthogonal and balanced, or at least very nearly so. Brand is the bin into which the linear factors are collected, and it becomes an actual attribute in the choice design. The right partition of the table shows the choice design. The `x1` attribute in the choice design is made from `x1`, `x4`, and `x7`, in the linear arrangement. These are the three size factors. Similarly, `x2` is made from `x2`, `x5`, and `x8`, in the linear arrangement. These are the three price factors. Finally, `x3` is made from the three top factors, `x3`, `x6`, and `x9`. This information is conveyed in the following table:

	x1	x2	x3
Brand 1	x1	x2	x3
Brand 2	x4	x5	x6
Brand 3	x7	x8	x9

This table provides the rules for constructing the choice design from the linear arrangement. We call this the *key* to constructing the choice design.

Blocking the Choice Design

The sports beverage example from Table 8 has 36 choice sets. Many choice designs are even larger. Even 36 choice sets might be too many judgments for one subject to make. Often larger designs are broken up into subsets or *blocks*. The number of blocks depends on the number of choice sets and the complexity of the choice task. For example, 36 choice sets might be small enough that no blocking is necessary, or instead, they can be divided into 2 blocks of size 18, 3 blocks of size 12, 4 blocks of size 9, 6 blocks of size 6, 9 blocks of size 4, 12 blocks of size 3, 18 blocks of size 2, or even 36 blocks of size 1. Technically, subjects *should* each see exactly one choice set. Showing subjects more than one choice set is economical, and in practice, most researchers almost always show multiple choice sets to each subject. The number of sets shown does not change the expected utilities, however, it does affect the covariance structure. Sometimes, attributes are highly correlated within blocks, particularly with small block sizes, but that is not a problem as long as they are not highly correlated over the entire design.

Efficiency of a Choice Design

All of the efficiency theory discussed so far concerns linear models ($\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$). In linear models, the parameter estimates $\hat{\boldsymbol{\beta}}$ have variances proportional to $(\mathbf{X}'\mathbf{X})^{-1}$. A choice model has the following form

$$p(c_i|C) = \frac{\exp(U(c_i))}{\sum_{j=1}^{m} \exp(U(c_j))} = \frac{\exp(\mathbf{x}_i\boldsymbol{\beta})}{\sum_{j=1}^{m} \exp(\mathbf{x}_j\boldsymbol{\beta})}$$

The probability that an individual will choose one of the m alternatives, c_i, from choice set C is a nonlinear function of $\mathbf{x}_i$, the vector of coded attributes, and $\boldsymbol{\beta}$, the vector of unknown parameters. The variances of the parameter estimates in the discrete choice multinomial logit model are given by

$$V(\hat{\beta}) = \left[\Sigma_{k=1}^{n} N \left[\frac{\Sigma_{j=1}^{m} \exp(x_j'\beta)x_j x_j'}{\Sigma_{j=1}^{m} \exp(x_j'\beta)} - \frac{(\Sigma_{j=1}^{m} \exp(x_j'\beta)x_j)(\Sigma_{j=1}^{m} \exp(x_j'\beta)x_j)'}{(\Sigma_{j=1}^{m} \exp(x_j'\beta))^2}\right]\right]^{-1}$$

with

m —	brands
n —	choice sets
N —	people
x_j —	the attributes of the *jth* alternative of the *kth* choice set

In the choice model, ideally we would like to pick $\mathbf{x}$'s that make this variance matrix "small." Unfortunately, we cannot do this unless we know $\boldsymbol{\beta}$, and if we knew $\boldsymbol{\beta}$, we would not need to do the experiment. However, in the chair example on pages 556–579, we see how to make an efficient choice design when we are willing to make assumptions about $\boldsymbol{\beta}$ other than $\boldsymbol{\beta} = \mathbf{0}$.

Because we do not know $\boldsymbol{\beta}$, we often create experimental designs for choice models using efficiency criteria for linear models. We make a good design for a linear model by picking $\mathbf{x}$'s that minimize a function of $(\mathbf{X}'\mathbf{X})^{-1}$ and then convert our linear arrangement into a choice design. Certain assumptions must be made before applying ordinary general-linear-model theory to problems in marketing research. The usual goal in linear modeling is to estimate parameters and test hypotheses about those parameters. Typically, independence and normality are assumed. In full-profile conjoint study, each subject rates all products and separate ordinary-least-squares analyses are run for each subject. This is not a standard general linear model; in particular, observations are not independent and normality cannot be assumed. Discrete choice models, which are nonlinear, are even more removed from the general linear model.

Marketing researchers often make the critical assumption that designs that are good for general linear models are also good designs for conjoint and discrete choice models. We also make this assumption. We assume that an efficient design for a linear model is a good design for the multinomial logit model used in discrete choice studies. We assume that if we create the linear arrangement (one row per choice set and all of the attributes of all of the alternatives comprise that row), and if we strive for linear-model efficiency (near balance and orthogonality), then we will have a good design for measuring the utility of each alternative and the contributions of the factors to that utility. When we construct choice designs in this way, our designs have two nice properties. 1) Each attribute level occurs equally often (or at least nearly equally often) for each attribute of each alternative across all choice sets. 2) Each attribute is independent of every other attribute (or at least nearly independent), both those in the same alternative and those in all of the other alternatives. The design techniques discussed in this book, based on the assumption that linear arrangement efficiency is a good surrogate for choice design goodness, have been used quite successfully in the field for many years.

In some of the examples, we use the `%MktEx` macro to create a linear arrangement, from which we construct our final choice design. This seems to be a good, safe strategy. It is a good strategy because it makes designs where all attributes, both within and between alternatives, are orthogonal or at least nearly so. It is safe in the sense that you have enough choice sets and collect the right information so that very complex models, including models with alternative-specific effects, availability effects, and cross-effects, can be fit. However, it is good to remember that when you run the `%MktEx` macro and you get an efficiency value, it corresponds to the linear arrangement (which does not correspond directly to any real model of interest), not the efficiency of the design in the context of the choice model. The linear model efficiency is a surrogate for the criterion of interest, the efficiency of the choice design, which is unknowable unless you know the parameters. Other examples optimize choice design efficiency based on an assumed parameter vector.

Coding, Efficiency, Balance, and Orthogonality

This section discusses coding in the context of linear models. However, the coding schemes used in this section apply equally to choice models. The next section builds on the materials discussed in this section and discusses coding in the context of a choice model.

Coding is the process or replacing our design factors by the set of indicator or coded variables that are actually used when the model is fit. We mention on page 63 that we use a special orthogonal coding of $\mathbf{X}$ (specifically, the standardized orthogonal contrast coding) when computing design efficiency. This section shows that coding and other codings. Even if you gloss over the mathematical details, this section is informative, because it provides insights into coding and the meaning of 100% efficiency and less than 100% efficient designs.

Table 9 displays the nonorthogonal less-than-full-rank *binary* or *indicator* codings for two-level through five-level factors. This is also known as "GLM coding" since it is the coding that is used by default in PROC GLM. It is requested with PROC TRANSREG as follows: `class(x / zero=none)`. In other procedures (such as PROC PHREG and PROC LOGISTIC), it is requested by specifying `param=glm`. There is one column for each level, and the coding contains a 1 when the level matches the column and a zero otherwise. We use these codings in many places throughout the examples.

Table 10 displays the nonorthogonal full-rank binary (or indicator or *reference cell* codings for two-level through five-level factors. This coding is like the full-rank coding shown previously, except that the column corresponding to the *reference level* has been dropped. It is requested with PROC TRANSREG as follows: `class(x)`. In other procedures (such as PROC PHREG and PROC LOGISTIC), it is requested by specifying `param=reference`. Frequently, the reference level is the last level, but it can be any level. We use these codings in many places throughout the examples.

Table 11 displays the nonorthogonal *effects coding* or *deviations from means coding* for two-level* through five-level factors. The effects coding differs from the full-rank binary coding in that the former always has a –1 to indicate the reference level. It is requested with PROC TRANSREG as follows: `class(x / effects)`. In other procedures (such as PROC PHREG and PROC LOGISTIC), it is requested by specifying `param=effect`. We use these codings in many places throughout the examples.

Table 12 displays the *orthogonal contrast coding* for two-level through five-level factors. They are the same as the orthogonal codings that are discussed in detail next, except that this version has not been scaled. Hence, all values are integers. It is requested with PROC TRANSREG as follows: `class(x / orthogonal)`. This coding is not available in most other procedures. Notice that the codings for each level form a contrast—the *ith* level versus all of the preceding levels and the last level.

Table 13 displays the *standardized orthogonal contrast coding*, for two-level through five-level factors, that the `%MktEx` macro and PROC OPTEX use internally. It is requested with PROC TRANSREG as follows: `class(x / standorth)` (or you can instead specify `class(x / ortheffect)`). In other procedures (such as PROC PHREG and PROC LOGISTIC), it is requested by specifying `param=ortheffect`. Notice that the sum of squares for the orthogonal coding of the two-level factor is 2. For both columns of the three-level factor, the sums of squares are 3; for the three columns of the four-level factor, the sums of squares are all 4; and for the four columns of the five-level factor, the sums of squares are all 5. For example, in the last column of the five-level factor, the sum of squares is: $-0.50^2 + -0.50^2 + -0.50^2 + 2^2 - 0.50^2 = 5$. Also notice that each column within a factor is orthogonal to all of the other columns—the sum of cross products is zero. For example, in the last two columns of the five-level factor, $-0.65 \times -0.5 + -0.65 \times -0.5 + 1.94 \times -.05 + 0 \times 2 + -0.65 \times -0.5 = 0$.

*The two-level effects coding is orthogonal, but the three-level and beyond codings are not.

Table 9
Nonorthogonal Less-Than-Full-Rank Binary or Indicator Coding

Two-Level

a	1	0
b	0	1

Three-Level

a	1	0	0
b	0	1	0
c	0	0	1

Four-Level

a	1	0	0	0
b	0	1	0	0
c	0	0	1	0
d	0	0	0	1

Five-Level

a	1	0	0	0	0
b	0	1	0	0	0
c	0	0	1	0	0
d	0	0	0	1	0
e	0	0	0	0	1

Table 10
Nonorthogonal Full-Rank Binary, Indicator, or Reference Cell Coding

Two-Level

a	1
b	0

Three-Level

a	1	0
b	0	1
c	0	0

Four-Level

a	1	0	0
b	0	1	0
c	0	0	1
d	0	0	0

Five-Level

a	1	0	0	0
b	0	1	0	0
c	0	0	1	0
d	0	0	0	1
e	0	0	0	0

Table 11
Nonorthogonal Effects or Deviations from Means Coding

Two-Level

a	1
b	-1

Three-Level

a	1	0
b	0	1
c	-1	-1

Four-Level

a	1	0	0
b	0	1	0
c	0	0	1
d	-1	-1	-1

Five-Level

a	1	0	0	0
b	0	1	0	0
c	0	0	1	0
d	0	0	0	1
e	-1	-1	-1	-1

Table 12
Orthogonal Contrast Coding

Two-Level

a	1
b	-1

Three-Level

a	1	-1
b	0	2
c	-1	-1

Four-Level

a	1	-1	-1
b	0	2	-1
c	0	0	3
d	-1	-1	-1

Five-Level

a	1	-1	-1	-1
b	0	2	-1	-1
c	0	0	3	-1
d	0	0	0	4
e	-1	-1	-1	-1

Table 13
Standardized Orthogonal Contrast Coding

Two-Level

a	1.00
b	-1.00

Three-Level

a	1.22	-0.71
b	0	1.41
c	-1.22	-0.71

Four-Level

a	1.41	-0.82	-0.58
b	0	1.63	-0.58
c	0	0	1.73
d	-1.41	-0.82	-0.58

Five-Level

a	1.58	-0.91	-0.65	-0.50
b	0	1.83	-0.65	-0.50
c	0	0	1.94	-0.50
d	0	0	0	2.00
e	-1.58	-0.91	-0.65	-0.50

Table 14

Choice Design				Coding							
				Brand	Brand	Brand	16	20		Twist	Twist
Brand	x1	x2	x3	1	2	3	oz.	oz.	Price	1	2
1	16 oz.	0.89	Twist Up	1	0	0	1	0	0.89	1	0
2	24 oz.	0.89	Twist Off	0	1	0	0	0	0.89	0	1
3	20 oz.	0.99	Pop Up	0	0	1	0	1	0.99	-1	-1
1	20 oz.	0.99	Pop Up	1	0	0	0	1	0.99	-1	-1
2	24 oz.	0.89	Twist Up	0	1	0	0	0	0.89	1	0
3	20 oz.	0.89	Twist Off	0	0	1	0	1	0.89	0	1
1	20 oz.	0.89	Twist Up	1	0	0	0	1	0.89	1	0
2	20 oz.	0.99	Twist Off	0	1	0	0	1	0.99	0	1
3	16 oz.	0.89	Twist Off	0	0	1	1	0	0.89	0	1
1	20 oz.	0.89	Twist Up	1	0	0	0	1	0.89	1	0
2	16 oz.	0.99	Twist Up	0	1	0	1	0	0.99	1	0
3	24 oz.	0.99	Pop Up	0	0	1	0	0	0.99	-1	-1
1	16 oz.	0.89	Twist Off	1	0	0	1	0	0.89	0	1
2	24 oz.	0.99	Pop Up	0	1	0	0	0	0.99	-1	-1
3	16 oz.	0.99	Twist Off	0	0	1	1	0	0.99	0	1
1	24 oz.	0.99	Twist Off	1	0	0	0	0	0.99	0	1
2	16 oz.	0.89	Twist Off	0	1	0	1	0	0.89	0	1
3	16 oz.	0.89	Pop Up	0	0	1	1	0	0.89	-1	-1

These codings are explained in more detail in the *SAS/STAT User's Guide*, PROC TRANSREG, DETAILS, "ANOVA Codings" section. All SAS documentation can be accessed online at: `http://support.sas.com`.

Table 14, using the design in Table 8, shows the less-than-full-rank binary coding (brand, 3 parameters), the full-rank binary coding (size, 2 parameters), and the effects coding (top, 2 parameters). Price (1 parameter) is not coded and instead is entered as is for a linear price effect.

All of the codings discussed in this section are equivalent to each other. They are equivalent in the sense that each is a transformation of the other. If $\mathbf{X_A}$ is a coded design matrix using one coding (or mix of codings), and $\mathbf{X_B}$ is coded from the same design matrix using some other coding (or mix of codings), then there exists a transformation matrix $\mathbf{T}$ such that $\mathbf{X_A} = \mathbf{X_B T}$ and $\mathbf{X_A T^{-1}} = \mathbf{X_B}$. Analyses using the different $\mathbf{X}$ matrices will produce the same predicted values, but different parameter estimates. You can always transform one set of parameters to another. This implies that we can create a design with one coding and analyze the data with one or more different codings and get equivalent results. **We will frequently use the standardized orthogonal contrast coding when creating designs and some other coding when analyzing the data.**

Note that the two orthogonal codings are both equivalent to a Gram-Schmidt orthogonalization of the effects coding, with a subsequent scaling of the columns to get the sums of squares right. You can run the following step to see this with a five-level factor:

```
proc iml;
   x = designf((1:5)‘);          /* X = effects coding,  A = X * inv(T)     */
   call gsorth(a,t,b,x);         /* A = Gram-Schmidt orthogonalization of X */
   print "Orthogonal Contrast Coding" /
         (a * inv(diag(a[1:ncol(a),])) * diag(1:ncol(a)));
   print "Standardized Orthogonal Contrast Coding" /
         (a * sqrt(nrow(a)));
   quit;
```

Recall that our measures of linear model design efficiency are scaled to range from 0 to 100.

$$A\text{-efficiency} = 100 \times \frac{1}{N_D \ \mathrm{trace}\left((\mathbf{X}'\mathbf{X})^{-1}\right)/p}$$

$$D\text{-efficiency} = 100 \times \frac{1}{N_D \ |(\mathbf{X}'\mathbf{X})^{-1}|^{1/p}}$$

When computing D-efficiency or A-efficiency, we code $\mathbf{X}$ so that when the design is orthogonal and balanced, $\mathbf{X}'\mathbf{X} = N_D\mathbf{I}$ where $\mathbf{I}$ is a $p \times p$ identity matrix. When our design is orthogonal and balanced, $(\mathbf{X}'\mathbf{X})^{-1} = \frac{1}{N_D}\mathbf{I}$, and $\mathrm{trace}\left((\mathbf{X}'\mathbf{X})^{-1}\right)/p = |(\mathbf{X}'\mathbf{X})^{-1}|^{1/p} = 1/N_D$. In this case, the two denominator terms cancel and efficiency is 100%. As the average variance increases, efficiency decreases.

The following statements show the coding of a 2×6 full-factorial design in 12 runs:

```
proc iml; /* orthogonal coding, levels must be 1, 2, ..., m */
   reset fuzz;

   start orthogcode(x);
      levels = max(x);
      xstar  = shape(x, levels - 1, nrow(x))‘;
      j = shape(1 : (levels - 1), nrow(x), levels - 1);
      r = sqrt(levels # (x / (x + 1))) # (j = xstar) -
          sqrt(levels / (j # (j + 1))) # (j > xstar | xstar = levels);
      return(r);
      finish;

   Design = (1:2)‘ @ j(6, 1, 1) || {1, 1} @ (1:6)‘;
   X = j(12, 1, 1) || orthogcode(design[,1]) || orthogcode(design[,2]);
   print design[format=1.]  '  '
         x[format=5.2 colname={'Int' 'Two' 'Six'} label=' '];

   XpX = x‘ * x;     print xpx[format=best5.];
   Inv = inv(xpx);   print inv[format=best5.];
   d_eff = 100 / (nrow(x) #   det(inv) ## (1 / ncol(inv)));
   a_eff = 100 / (nrow(x) # trace(inv)       / ncol(inv));
   print    'D-efficiency =' d_eff[format=6.2 label=' ']
         '   A-efficiency =' a_eff[format=6.2 label=' '];
```

The orthogonal coding function requires the factor levels to be consecutive positive integers beginning with one and ending with m for an m-level factor. Note that the IML operator # performs ordinary (scalar) multiplication, and ## performs exponentiation. The results are as follows:

```
               Design        Int   Two   Six

                    1 1     1.00  1.00  1.73 -1.00 -0.71 -0.55 -0.45
                    1 2     1.00  1.00  0.00  2.00 -0.71 -0.55 -0.45
                    1 3     1.00  1.00  0.00  0.00  2.12 -0.55 -0.45
                    1 4     1.00  1.00  0.00  0.00  0.00  2.19 -0.45
                    1 5     1.00  1.00  0.00  0.00  0.00  0.00  2.24
                    1 6     1.00  1.00 -1.73 -1.00 -0.71 -0.55 -0.45
                    2 1     1.00 -1.00  1.73 -1.00 -0.71 -0.55 -0.45
                    2 2     1.00 -1.00  0.00  2.00 -0.71 -0.55 -0.45
                    2 3     1.00 -1.00  0.00  0.00  2.12 -0.55 -0.45
                    2 4     1.00 -1.00  0.00  0.00  0.00  2.19 -0.45
                    2 5     1.00 -1.00  0.00  0.00  0.00  0.00  2.24
                    2 6     1.00 -1.00 -1.73 -1.00 -0.71 -0.55 -0.45

                                          XpX

                       12     0     0     0     0     0     0
                        0    12     0     0     0     0     0
                        0     0    12     0     0     0     0
                        0     0     0    12     0     0     0
                        0     0     0     0    12     0     0
                        0     0     0     0     0    12     0
                        0     0     0     0     0     0    12

                                          Inv

                    0.083     0     0     0     0     0     0
                        0 0.083     0     0     0     0     0
                        0     0 0.083     0     0     0     0
                        0     0     0 0.083     0     0     0
                        0     0     0     0 0.083     0     0
                        0     0     0     0     0 0.083     0
                        0     0     0     0     0     0 0.083

                   D-efficiency = 100.00    A-efficiency = 100.00
```

The following statements are a continuation of the preceding program and compute *D*-efficiency and *A*-efficiency for just a subset of the design (the first 10 rows):

```
design = design[1:10,];
x = j(10, 1, 1) || orthogcode(design[,1]) || orthogcode(design[,2]);
inv = inv(x‘ * x);
d_eff = 100 / (nrow(x) #   det(inv) ## (1 / ncol(inv)));
a_eff = 100 / (nrow(x) # trace(inv)       / ncol(inv));
print    ’D-efficiency =’ d_eff[format=6.2 label=’ ’]
      ’   A-efficiency =’ a_eff[format=6.2 label=’ ’];
quit;
```

With the full orthogonal and balanced design, $\mathbf{X'X} = N_D\mathbf{I} = 12\mathbf{I}$, which means $(\mathbf{X'X})^{-1} = \frac{1}{N_D}\mathbf{I} = \frac{1}{12}\mathbf{I}$, and D-efficiency = 100%. With a nonorthogonal design (for example, with the first 10 rows of the 2×6 full-factorial design), D-efficiency and A-efficiency are less than 100%. The results are as follows:

```
                    D-efficiency =  92.90    A-efficiency =  84.00
```

In this case, $|(\mathbf{X'X})^{-1}|^{1/p}$ and $\text{trace}\,((\mathbf{X'X})^{-1})/p$ are multiplied in the denominator of the efficiency formulas by $\frac{1}{N_D} = \frac{1}{10}$. If an orthogonal and balanced design were available for this problem, then $(\mathbf{X'X})^{-1}$ would equal $\frac{1}{N_D}\mathbf{I} = \frac{1}{10}\mathbf{I}$. Since an orthogonal and balanced design is not possible (6 does not divide 10), both D-efficiency and A-efficiency are less than 100%, even with the optimal design. An orthogonal and balanced design for a main-effects model, with a variance matrix equal to $\frac{1}{N_D}\mathbf{I}$, is the standard by which 100% efficiency is gauged, even when we know such a design cannot exist. The standard is the average variance for the maximally efficient *potentially hypothetical* design, which is knowable, not the average variance for the optimal design, which for many problems we have no way of knowing.

For our purposes in this book, we only consider experimental designs with at least as many runs as parameters. A *saturated* or *tight* design has as many runs as there are parameters.[†] The number of parameters in a main-effects model is the sum of the numbers of levels of all of the factors, minus the number of factors, plus 1 for the intercept. Equivalently, since there are $m-1$ parameters in an m-level factor, the number of parameters is $1 + \sum_{j=1}^{k}(m_j - 1)$ for k factors, each with m_j levels.

If a main-effects design is orthogonal and balanced, then the design must be at least as large as the saturated design and the number of runs must be divisible by the number of levels of all the factors and by the products of the number of levels of all pairs of factors. For example, a $2 \times 2 \times 3 \times 3 \times 3$ design cannot be orthogonal and balanced unless the number of runs is divisible by 2 (twice because there are two 2's), 3 (three times because there are three 3's), $2 \times 2 = 4$ (once, because there is one pair of 2's), $2 \times 3 = 6$ (six times, two 2's times three 3's), and $3 \times 3 = 9$ (three times, three pairs of 3's).[*] If the design is orthogonal and balanced, then all of the divisions work without a remainder. However, all of the divisions working is a necessary but not sufficient condition for the existence of an orthogonal and balanced design. For example, 45 is divisible by 3 and $3 \times 3 = 9$, but an orthogonal and balanced saturated design 3^{22} (22 three-level factors) in 45 runs does not exist.

Coding and Reference Levels: The ZERO= Option

In this book, we do most of our coding using PROC TRANSREG and its MODEL statement. In some cases, such as full-profile conjoint analysis, we call PROC TRANSREG directly to perform the analysis. In other cases, such as coding the design before fitting a choice model, we call PROC TRANSREG directly to perform the coding, but we use PROC PHREG for the analysis. In still other cases, such as with the `%ChoicEff` macro, we specify PROC TRANSREG syntax, but do not call the procedure directly. The macro calls the procedure for us. No matter how it gets called, we often need to control the reference level or suppress the creation of a reference level for a less-than-full-rank coding. This

[†]This definition is in the context of a linear model. In a choice model, substitute number of choice sets times the number of alternatives minus one for the number of runs.

[*]In practice, we never have to do any of these calculations ourselves, since they are done for us by the `%MktRuns` macro.

section describes the syntax involved in the PROC TRANSREG `zero=` option, which is used to control the reference level. In this section, only the relevant MODEL statement fragments are provided rather than the full statement. This section is important because this option is used a lot in this book, and while it is not a difficult option, you might find it confusing if you do not have the background provided here.

By default, when coding, an indicator variable is not created for the last level and hence a coefficient is not computed for that level. This is called *reference-cell coding.* For example, by default, `x` with values 1, 2, and 3 is coded as follows:

`x`	Coding	
1	1	0
2	0	1
3	0	0

The level '3' is the reference level. You could instead create an indicator variable for all levels as follows:

`x`	Coding		
1	1	0	0
2	0	1	0
3	0	0	1

However, in many modeling situations, such as ANOVA models, regression models with intercepts, and choice models, the coefficient for the indicator variable for the last level would be zero, because that last column is redundant given all of the columns that come before. The name of the option, `zero=`, comes from the idea that this option lets you specify the level that will have a structural zero coefficient. There are several ways that we use the zero= option:

`zero=first`
specifies the first level as the reference level for all factors in the `class` specification in which it is applied. Coded variables are created for all but the first level for each factor.

`zero=last`
specifies the last level as the reference level for all factors in the `class` specification in which it is applied. Coded variables are created for all but the last level for each factor. This is the default.

`zero=none`
does not create a reference level for any of the factors in the `class` specification in which it is applied. Indicator variables are created for all levels of all factors.

`zero=sum`
does not create a reference level for any of the factors in the `class` specification in which it is applied. Indicator variables are created for all levels of all factors. The parameter estimates are constrained to sum to zero. This option is useful for full-profile conjoint analysis but not for choice modeling. This is because it can only be used when PROC TRANSREG is doing the analysis, not when PROC TRANSREG is just doing the coding.

`zero=`*formatted-value-list*
specifies the level corresponding to the formatted value as the reference level for all factors in the `class` specification in which it is applied. Coded variables are created for all but the specified level for each factor. The first formatted value applies to the first factor, the second formatted value applies to the second factor, and so on. When the formatted value list is shorter than the factor list, the default `zero=last` is used for the remaining factors. The formatted values must appear in quotes.

The remainder of this section illustrates properties of the `zero=` option. We assume that `x1-x6` are all three level factors with levels 1, 2, and 3. Also assume that the first three rows of the design are as follows:

x1	x2	x3	x4	x5	x6
1	1	1	1	1	1
2	2	2	2	2	2
3	3	3	3	3	3

Note that you cannot mix `zero=none`, `zero=first`, or `zero=last` with `zero=`*formatted-value-list* within a single `class` specification. However, you can use different options if you use multiple `class` specifications, as is the following example:

```
class(x1 x2 / zero=none) class(x3 x4 / zero=first) class(x5 x6 / zero=’2’ ’3’)
```

The first three rows of the coded design are as follows:

x11	x12	x13	x21	x22	x23	x32	x33	x42	x43	x51	x53	x61	x62
1	0	0	1	0	0	0	0	0	0	1	0	1	0
0	1	0	0	1	0	1	0	1	0	0	0	0	1
0	0	1	0	0	1	0	1	0	1	0	1	0	0

In this coded design, the columns are labeled by the name of the factor (from `x1-x6`) and formatted value of the relevant level (1, 2, or 3). For example, `x11` is the indicator variable for the first level of `x1`.

With `zero=`*formatted-value-list*, you can achieve the same effect as `zero=first` by specifying the first formatted value and the same effect as `zero=last` by specifying the last formatted value. Additionally, you can achieve the same effect as `zero=none` by specifying a nonexistent level such as blank. The following specification is equivalent to the preceding specification and produces the same coded design:

```
class(x1-x6 / zero=’ ’  ’ ’  ’1’  ’1’  ’2’  ’3’)
```

PROC TRANSREG warns you when you specify a nonexistent reference level as follows:

```
WARNING: Reference level ZERO=’’ was not found for variable x1.
```

This warning can be safely ignored because the specification was deliberate.

Sometimes, it is useful to specify `zero=none` for the first factor and use the default `zero=last` for the remaining factors. The following illustrates one way of doing this:

```
class(x1 / zero=none) class(x2-x6)
```

The first three rows of the coded design are as follows:

x11	x12	x13	x21	x22	x31	x32	x41	x42	x51	x52	x61	x62
1	0	0	1	0	1	0	1	0	1	0	1	0
0	1	0	0	1	0	1	0	1	0	1	0	1
0	0	1	0	0	0	0	0	0	0	0	0	0

The following illustrates another way of doing this same thing:

```
class(x1-x6 / zero=’ ’)
```

The results are the same as before. Note that the nonexistent blank level applies to `x1`. Since the `zero=` list is exhausted, the reference level for the remaining factors, `x2-x6`, is the default last level.

In choice modeling, the brand factor might have a level labeled as 'None' for no purchase or none of the above. You can specify it in quotes in the `zero=` option as you would any other level. The following illustrates this:

```
   class(Brand Price / zero='None')
```

This creates indicator variables for all but the 'None' level of `Brand` and for all but the last level of `Price`. Note that this is quite different from the following:

```
   class(Brand Price / zero=None)
```

This specification creates indicator variables for every level of both factors.

The next section discusses coding and design efficiency. The section after that gives some more detail about the `zero=` option and its effects on coding.

Coding and the Efficiency of a Choice Design

The previous sections discuss several types of coding including reference cell coding, less-than-full-rank coding, effects coding, and the standardized orthogonal contrast coding. This section discusses their use in choice modeling. This section is important because it provides you with some context you can use to evaluate the goodness of your choice design.

Before you use a choice design, you should code it, compute the variance-covariance matrix of the parameter estimates, and check the design efficiency. You can do this with the `%ChoicEff` macro. It uses PROC TRANSREG to do the coding, and you specify the PROC TRANSREG syntax and options that control the coding. When you code the design properly (at least for some designs), you can get a relative *D*-efficiency on a 0 to 100 scale. This is done by using a standardized orthogonal contrast coding and scaling the observed *D*-efficiency relative to the *D*-efficiency for a possibly hypothetical optimal design. In a linear model, the variance matrix for the potentially hypothetical optimal design is $\frac{1}{N_D}\mathbf{I}$ where N_D is the number or runs in the linear arrangement. Similarly, in a choice model, the variance matrix for the potentially hypothetical optimal design is in some cases $\frac{1}{N_S}\mathbf{I}$ where N_S is the number of choice sets. When a choice design is made from a linear arrangement using the method shown previously in this chapter, $N_D = N_S$.

To illustrate, consider a choice design with 3 three-level attributes, three alternatives, and three choice sets. This next example constructs an optimal generic choice design for this specification. Later sections discuss syntax and designs like this one in more detail. This section concentrates on results—specifically design efficiency and the variances of the parameter estimates. This design is constructed and evaluated as follows:

```
   %mktex(3 ** 4,                       /* set number and factor levels        */
          n=9)                          /* num of sets times num of alts (3x3)  */

   %mktlab(data=design,                 /* design from MktEx                   */
           vars=Set x1-x3)              /* new variable names                  */
```

```
%choiceff(data=final,              /* candidate set of choice sets         */
          init=final(keep=set),    /* select these sets from candidates    */
          model=class(x1-x3 / sta), /* model with stdzd orthogonal coding  */
          nsets=3,                 /* 3 choice sets                        */
          nalts=3,                 /* 3 alternatives per set               */
          options=relative,        /* display relative D-efficiency        */
          beta=zero)               /* assumed beta vector, Ho: b=0         */

proc print data=bestcov label;     /* covariance matrix from ChoicEff      */
   title 'Variance-Covariance Matrix';
   id __label;
   label __label = '00'x;          /* hex null suppress label header       */
   var x:;
   run;

title;
```

The `sta`* (short for `standorth`) option in the `model=` option is new with SAS 9.2 and requests a standardized orthogonal contrast coding. We use it whenever possible, because in some situations, it gives us a relative *D*-efficiency in the range 0 to 100. That is, for certain optimal designs such as this one, relative *D*-efficiency is 100 when the standardized orthogonal contrast coding is used. For other optimal designs, such as those with constraints such as a constant alternative, the maximum relative *D*-efficiency is less than 100 with this coding. The precise maximum is hard to know in general when there are constraints. With other codings, the raw *D*-efficiency can be any nonnegative value. In some cases, it can even be greater than 100.

A subset of the results are as follows:

```
                              Final Results

                        Design                  1
                        Choice Sets             3
                        Alternatives            3
                        Parameters              6
                        Maximum Parameters      6
                        D-Efficiency       3.0000
                        Relative D-Eff   100.0000
                        D-Error            0.3333
                        1 / Choice Sets    0.3333
```

*This option is first available with SAS 9.2. It will not be recognized, and it will cause an error in earlier SAS releases.

```
               Variable                                        Standard
          n      Name       Label     Variance     DF          Error

          1      x11        x1 1       0.33333      1        0.57735
          2      x12        x1 2       0.33333      1        0.57735
          3      x21        x2 1       0.33333      1        0.57735
          4      x22        x2 2       0.33333      1        0.57735
          5      x31        x3 1       0.33333      1        0.57735
          6      x32        x3 2       0.33333      1        0.57735
                                                   ==
                                                    6

                         Variance-Covariance Matrix

              x1 1        x1 2        x2 1        x2 2        x3 1        x3 2

  x1 1     0.33333     0.00000     0.00000     0.00000     0.00000     0.00000
  x1 2     0.00000     0.33333     0.00000     0.00000     0.00000     0.00000
  x2 1     0.00000     0.00000     0.33333     0.00000     0.00000     0.00000
  x2 2     0.00000     0.00000     0.00000     0.33333     0.00000     0.00000
  x3 1     0.00000     0.00000     0.00000     0.00000     0.33333     0.00000
  x3 2     0.00000     0.00000     0.00000     0.00000     0.00000     0.33333
```

Starting with the last table, the variance-covariance matrix is $\frac{1}{3}\mathbf{I}$. The 3 in the denominator comes from the fact that there are 3 choice sets. For reference, the last line in the first table displays one over the number of choice sets. Above that, it displays D-error (1/3), which is the inverse of D-efficiency (3, the number of choice sets). Relative D-efficiency is equal to D-efficiency divided by the optimal value (the number of choice sets) and multiplied by 100. In an optimal design such as this one, relative D-efficiency is 100. The middle table displays the variances, which are the diagonal values from the variance-covariance matrix, and the standard errors, which are the square roots of the variances. With the standardized orthogonal contrast coding used here, an optimal design has all zeros on the off diagonals and $1/N_S$ on the diagonal.

It is also good to compare the number of parameters in the model with the maximum number of parameters that could be estimated with the number of choice sets and alternatives in this experiment. These are shown in the first table. In this case, both of these numbers are the same (6) showing that the design is saturated. Unless you have an optimal design such as this one, you will usually not want the number of parameters to be this close to the maximum. The maximum value is the number of choice sets times the number of alternatives minus one: $3(3-1) = 6$.

In practice, most of our designs are not optimal like this one. However, you can still use the standardized orthogonal contrast coding to see how much bigger the variances are and use that information to guide your design construction decisions. To illustrate, consider the next example, which creates a random choice design. This is *not* a recommended strategy; it is just done to show how a less-than-optimal design compares to the optimal design. The following steps make and evaluate a random design:

```
data final;                          /* random design                         */
   do Set = 1 to 3;                  /* 3 choice sets                         */
      do Alt = 1 to 3;               /* 3 alternatives                        */
         x1 = ceil(3 * uniform(151));/* random levels for each attr           */
         x2 = ceil(3 * uniform(151));
         x3 = ceil(3 * uniform(151));
         output;
         end;
      end;
   run;

%choiceff(data=final,               /* candidate set of choice sets          */
          init=final(keep=set),     /* select these sets from candidates     */
          model=class(x1-x3 / sta), /* model with stdzd orthogonal coding    */
          nsets=3,                  /* 3 choice sets                         */
          nalts=3,                  /* 3 alternatives per set                */
          options=relative,         /* display relative D-efficiency         */
          beta=zero)                /* assumed beta vector, Ho: b=0          */

proc print data=bestcov label;      /* covariance matrix from ChoicEff       */
   title 'Variance-Covariance Matrix';
   id __label;
   label __label = '00'x;           /* hex null suppress label header        */
   var x:;
   run;

title;
```

The same tables as in the previous example are as follows:

```
                         Final Results

                  Design                    1
                  Choice Sets               3
                  Alternatives              3
                  Parameters                6
                  Maximum Parameters        6
                  D-Efficiency         1.0000
                  Relative D-Eff      33.3333
                  D-Error              1.0000
                  1 / Choice Sets      0.3333
```

```
              Variable                                      Standard
         n      Name       Label     Variance     DF         Error

         1      x11        x1 1       1.00000      1       1.00000
         2      x12        x1 2       2.33333      1       1.52753
         3      x21        x2 1       2.00000      1       1.41421
         4      x22        x2 2       2.66667      1       1.63299
         5      x31        x3 1       3.00000      1       1.73205
         6      x32        x3 2       1.00000      1       1.00000
                                                  ==
                                                   6

                           Variance-Covariance Matrix

             x1 1         x1 2         x2 1         x2 2         x3 1         x3 2

 x1 1     1.00000      0.00000      1.00000      1.15470     -1.00000     -0.57735
 x1 2     0.00000      2.33333     -0.00000      1.00000      0.57735      0.33333
 x2 1     1.00000     -0.00000      2.00000      1.73205     -0.50000     -0.86603
 x2 2     1.15470      1.00000      1.73205      2.66667     -0.86603     -0.83333
 x3 1    -1.00000      0.57735     -0.50000     -0.86603      3.00000      0.57735
 x3 2    -0.57735      0.33333     -0.86603     -0.83333      0.57735      1.00000
```

Now our variances range from 1 to 3 instead of all being 1/3. The design is 33.3333% as efficient as the optimal design. Even if we did not have the results from the optimal design for reference, we can see from the table that one over the number of choice sets = 0.3333, so our variances are way bigger than we would expect.

Even if you plan on using a different coding for the analysis, it is good to evaluate the design using the standardized orthogonal contrast coding and see how large the variances are relative to the optimal value. Note, however, that in many situations, we do not know what the optimal variance is. One over the number of choice sets might be too small when the design is more complicated than an optimal generic choice design. To illustrate, again consider the same problem, but this time we will force the third alternative to be constant (all levels 2). The following steps create and evaluate the design:

```
%mktex(3 ** 3,                     /* just the factor levels            */
       n=27)                       /* number of candidate alts          */

%mktlab(data=design,               /* design from MktEx                 */
        int=f1-f3)                 /* flag which alt can go where, 3 alts  */

data final;                        /* all candidates go to alt 1 and 2     */
   set final;                      /* x1=2 x2=2 x3=2 also goes to alt 3    */
   f3 = (x1 eq 2 and x2 eq 2 and x3 eq 2);
   run;
```

```
%choiceff(data=final,              /* candidate set of alternatives       */
          model=class(x1-x3 / sta), /* model with stdzd orthogonal coding  */
          seed=205,                /* random number seed                  */
          nsets=3,                 /* 3 choice sets                       */
          flags=f1-f3,             /* flag which of the 3 alts go where   */
          options=relative,        /* display relative D-efficiency       */
          beta=zero)               /* assumed beta vector, Ho: b=0        */

proc print data=bestcov label;     /* covariance matrix from ChoicEff     */
   title 'Variance-Covariance Matrix';
   id __label;
   label __label = '00'x;          /* hex null suppress label header      */
   var x:;
   run;

title;
```

The same tables as in the previous examples are as follows:

```
                              Final Results

                        Design                 1
                        Choice Sets            3
                        Alternatives           3
                        Parameters             6
                        Maximum Parameters     6
                        D-Efficiency      1.5874
                        Relative D-Eff   52.9134
                        D-Error           0.6300
                        1 / Choice Sets   0.3333

                Variable                                  Standard
           n      Name      Label     Variance    DF        Error

           1      x11       x1 1       1.75000     1      1.32288
           2      x12       x1 2       1.08333     1      1.04083
           3      x21       x2 1       0.50000     1      0.70711
           4      x22       x2 2       0.66667     1      0.81650
           5      x31       x3 1       0.50000     1      0.70711
           6      x32       x3 2       0.66667     1      0.81650
                                                  ==
                                                   6
                              Variance-Covariance Matrix

                 x1 1        x1 2        x2 1        x2 2        x3 1        x3 2

      x1 1     1.75000    -0.28868    -0.12500     0.21651     0.12500     0.21651
      x1 2    -0.28868     1.08333    -0.07217    -0.37500     0.07217    -0.37500
      x2 1    -0.12500    -0.07217     0.50000    -0.14434     0.00000     0.14434
      x2 2     0.21651    -0.37500    -0.14434     0.66667    -0.14434    -0.16667
      x3 1     0.12500     0.07217     0.00000    -0.14434     0.50000     0.14434
      x3 2     0.21651    -0.37500     0.14434    -0.16667     0.14434     0.66667
```

This is a small and easy problem for the %ChoicEff macro, and it is given all possible candidates from which to work. Hence, it has almost certainly found the optimal design for this specification. However, relative *D*-efficiency is 52.9134% and the variances are all larger than one over the number of choice sets.*

These results show that for this small design, it appears that the maximum *D*-efficiency is 1.5874. If you know the maximum possible *D*-efficiency (or even have a guess), you can use it as a scaling factor for relative *D*-efficiency instead of using the default (the number of choice sets). The following step specifies the maximum *D*-Efficiency in the rscale= (relative efficiency scaling factor) option:

```
%choiceff(data=final,                 /* candidate set of alternatives       */
          model=class(x1-x3 / sta),   /* model with stdzd orthogonal coding  */
          seed=205,                   /* random number seed                  */
          nsets=3,                    /* 3 choice sets                       */
          flags=f1-f3,                /* flag which of the 3 alts go where   */
          rscale=1.5874,              /* scale using previous D-efficiency   */
          options=relative,           /* display relative D-efficiency       */
          beta=zero)                  /* assumed beta vector, Ho: b=0        */

proc print data=bestcov label;        /* covariance matrix from ChoicEff     */
   title 'Variance-Covariance Matrix';
   id __label;
   label __label = '00'x;             /* hex null suppress label header      */
   var x:;
   run;

title;
```

The same tables as in the previous examples are as follows:

*If you run this step repeatedly with different seeds, you will get the same relative *D*-efficiency, but the individual variances will change as the %ChoicEff finds different but equivalent designs.

```
                          Final Results

                    Design                   1
                    Choice Sets              3
                    Alternatives             3
                    Parameters               6
                    Maximum Parameters       6
                    D-Efficiency        1.5874
                    Relative D-Eff    100.0001
                    D-Error             0.6300
                    1 / Choice Sets     0.3333

            Variable                                       Standard
     n        Name       Label     Variance     DF          Error

     1        x11        x1 1       1.75000      1        1.32288
     2        x12        x1 2       1.08333      1        1.04083
     3        x21        x2 1       0.50000      1        0.70711
     4        x22        x2 2       0.66667      1        0.81650
     5        x31        x3 1       0.50000      1        0.70711
     6        x32        x3 2       0.66667      1        0.81650
                                                ==
                                                 6

                      Variance-Covariance Matrix

              x1 1        x1 2        x2 1        x2 2        x3 1        x3 2

x1 1       1.75000    -0.28868    -0.12500     0.21651     0.12500     0.21651
x1 2      -0.28868     1.08333    -0.07217    -0.37500     0.07217    -0.37500
x2 1      -0.12500    -0.07217     0.50000    -0.14434     0.00000     0.14434
x2 2       0.21651    -0.37500    -0.14434     0.66667    -0.14434    -0.16667
x3 1       0.12500     0.07217     0.00000    -0.14434     0.50000     0.14434
x3 2       0.21651    -0.37500     0.14434    -0.16667     0.14434     0.66667
```

Now, relative *D*-efficiency is approximately 100. It would be exactly 100 if it there were no rounding error in the *D*-efficiency displayed in the first table. The variances and covariances are unchanged.

In summary, the `%ChoicEff` macro gives you options and context to help you evaluate your choice designs. In some simple situations, the maximum *D*-efficiency is clear, and it can be used to scale the current design *D*-efficiency to get a relative *D*-efficiency on a 0 to 100 scale. When the maximum is not clear, you can instead specify your own scale factor.

Orthogonal Coding and the ZERO=' ' Option

The zero= option was explained in a preceding section. There is one more aspect of zero=' ' versus zero=none usage that needs to be explained. The zero=none option is designed to transform binary (0,1) reference cell coding (one indicator variable for every level except one) into a cell-means coding (one indicator variable for every level). It is not designed for use with the effects (also known as deviations from means) coding (specified with effects, eff, deviations, dev), or the orthogonal codings (specified with orthogonal, ort, standorth, sta). However, you an use zero=' ' to specify a cell-means-style coding for just the first factor in a class specification with any of those options. This lets you specify these codings within groups. This is probably used most often when evaluating a choice design that has alternative-specific effects. To illustrate, consider the following data set:

```
data x;
   input Brand $ x1-x2;
   datalines;
A 1 1
A 1 2
A 2 1
A 2 2
B 1 1
B 1 2
B 2 1
B 2 2
C 1 1
C 1 2
C 2 1
C 2 2
;
```

The following PROC TRANSREG step codes this design using the standardized orthogonal contrast coding for alternative-specific effects within brand:

```
proc transreg design data=x;
   model class(brand / lprefix=0)
         class(brand * x1 brand * x2 / sta zero=' ' lprefix=0 2 2);
   output out=coded(drop=_: in:) separators='' ' ';
   run;

proc print label noobs; run;
```

Note that zero=' ' applies to only the first factor in that class specification (Brand), and it applies to it every place that it is used in that class specification. The results are as follows:

```
                  A      B      C      A      B      C
   A     B      x11    x11    x11    x21    x21    x21    Brand    x1    x2

   1     0       1      0      0      1      0      0       A       1     1
   1     0       1      0      0     -1      0      0       A       1     2
   1     0      -1      0      0      1      0      0       A       2     1
   1     0      -1      0      0     -1      0      0       A       2     2
   0     1       0      1      0      0      1      0       B       1     1
   0     1       0      1      0      0     -1      0       B       1     2
   0     1       0     -1      0      0      1      0       B       2     1
   0     1       0     -1      0      0     -1      0       B       2     2
   0     0       0      0      1      0      0      1       C       1     1
   0     0       0      0      1      0      0     -1       C       1     2
   0     0       0      0     -1      0      0      1       C       2     1
   0     0       0      0     -1      0      0     -1       C       2     2
```

The first 8 columns contain the coded design, and the last 3 contain the input factors (`class` variables). The specification `class(brand / lprefix=0)` creates the first two columns, labeled `A` and `B`, which are indicator variables for the first two brands. The third brand, `C`, corresponds to the reference level. For all three brands, both `x1` and `x2` are coded with a standardized orthogonal contrast coding within brand. Since both `x1` and `x2` have only two levels, there is only one coded variable for each factor for each brand. The interaction between `Brand` and `x1` creates the design columns labeled as `A x11`, `B x11`, and `C x11`, and the interaction between `Brand` and `x2` creates the design columns labeled as `A x21`, `B x21`, and `C x21`.

You can better understand why the columns `A x11` through `C x21` are coded as they are by examining the main effects that go into creating these interaction terms. The following steps create and display both the main effects and interactions:

```
proc transreg design data=x;
   model class(brand | x1 brand | x2 / sta zero=’ ’ lprefix=0 2 2);
   output out=coded separators=’’ ’ ’;
   run;

proc print label noobs;
   var BrandA BrandB BrandC x11 x21 BrandAx11 BrandBx11 BrandCx11
       BrandAx21 BrandBx21 BrandCx21;
   run;
```

The results are as follows:

```
                                     A      B      C      A      B      C
 A    B    C    x11    x21    x11    x11    x11    x21    x21    x21

 1    0    0      1      1      1      0      0      1      0      0
 1    0    0      1     -1      1      0      0     -1      0      0
 1    0    0     -1      1     -1      0      0      1      0      0
 1    0    0     -1     -1     -1      0      0     -1      0      0
 0    1    0      1      1      0      1      0      0      1      0
 0    1    0      1     -1      0      1      0      0     -1      0
 0    1    0     -1      1      0     -1      0      0      1      0
 0    1    0     -1     -1      0     -1      0      0     -1      0
 0    0    1      1      1      0      0      1      0      0      1
 0    0    1      1     -1      0      0      1      0      0     -1
 0    0    1     -1      1      0      0     -1      0      0      1
 0    0    1     -1     -1      0      0     -1      0      0     -1
```

The columns labeled `A x11` through `C x21` are the same as we saw before. `A x11` is the element-wise product of `A` and `x11`, `B x11` is the element-wise product of `B` and `x11`, `C x11` is the element-wise product of `C` and `x11`, `A x21` is the element-wise product of `A` and `x21`, `B x21` is the element-wise product of `B` and `x21`, and `C x21` is the element-wise product of `C` and `x21`. More information about the standardized orthogonal contrast coding, along with the use of `zero=' '`, can be found in the example starting on page 858.

Orthogonally Coding Price and Other Quantitative Attributes

For inherently quantitative factors such as price, you might want to use different strategies for coding during the analysis instead of using indicator variables or effects coding. When we create a design with a quantitative factor such as price, we do not have to do anything special. The orthogonal coding what we use to make qualitative factors is just as applicable when the factor are quantitative. See page 251 for more information. However, for analysis, we might want a different coding than the binary or effects coding. For example, imagine a choice experiment involving SUV's with price as an attribute and with levels of $27,500, $30,000, and $32,500. You probably will not code them as is and just add these prices directly to the model, because these values are considerably larger than the other values in your coded factors, which usually consist of values such as –1, 0, and 1. You might believe that choice is not a linear function of price; it might be nonlinear or quadratic. Hence, you might think about adding a price-squared term, but squaring values this large is almost certain to cause problems with collinearity. When you are dealing with factors such as this, you are usually better off recoding them in a "nicer" way. The following table shows some of the steps in the recoding:

Price	Centered Price	Divide By Increment	Square
27,500	$27,500 - 30,000 = -2,500$	$-2,500/2,500 = -1$	$-1^2 = 1$
30,000	$30,000 - 30,000 = 0$	$0/2,500 = 0$	$0^2 = 0$
32,500	$32,500 - 30,000 = 2,500$	$2,500/2,500 = 1$	$1^2 = 1$

The second column shows the results of centering the values—subtracting the mean price of $30,000. The third column shows the results of dividing the centered values by the increment between values, 2,500. The fourth column shows the square of the third column. These last two columns make much better linear and quadratic price terms than the original price and the original price squared, however, we can do better still. The next table shows the final steps and the full, orthogonal, quadratic coding.

Coding			Centered Quadratic					Multiply Through					Orthogonal Coding		
1	−1	1	1	−	2/3	=	1/3	3	×	1/3	=	1	1	−1	1
1	0	0	0	−	2/3	=	−2/3	3	×	−2/3	=	−2	1	0	−2
1	1	1	1	−	2/3	=	1/3	3	×	1/3	=	1	1	1	1

The first coding consists of an intercept, a linear term, and a quadratic term. Notice that the sum of the quadratic term is not zero, so the quadratic term is not orthogonal to the intercept. We can correct this by centering (subtracting the mean which is 2/3). After centering, all three columns are orthogonal. We can make the coding nicer still by multiplying the quadratic term by 3 to get rid of the fractions. The full orthogonal coding is shown in the last set of columns. Note, however, that only the last two columns are used. The intercept is just there to more clearly show that all columns are orthogonal. This orthogonal coding works for any three-level quantitative factor with equal intervals between adjacent levels.

For four equally-spaced levels, and with less detail, the linear and quadratic coding is shown in the last two columns of the following table:

Price	Center	Divide	Integers	Square	Center	Smallest Integers	Coding	
27500	−3750	−1.5	−3	9	4	1	−3	1
30000	−1250	−0.5	−1	1	−4	−1	−1	−1
32500	1250	0.5	1	1	−4	−1	1	−1
35000	3750	1.5	3	9	4	1	3	1

The Number of Factor Levels

The number of levels of the factors can affect design efficiency. Since two points define a line, it is inefficient to use more than two points to model a linear function. When a quadratic function is used (x and x^2 are included in the model), three points are needed—the two extremes and the midpoint. Similarly, four points are needed for a cubic function. More levels are needed when the functional form is unknown. Extra levels let you examine complicated nonlinear functions, with a cost of decreased efficiency for the simpler functions. When you assume that the function is linear, experimental points should not be spread throughout the range of experimentation.

We are often tempted to have more levels than we really need, particularly for factors such as price. If you expect to model a quadratic price function, you only need three price points. It might make sense to have one or two more price points so that you can test for departures from the quadratic model, but you do not want more than that. You probably would never be interested in a price function more complicated than a cubic function. Creating a design with many price points and then fitting a low-order price function reduces efficiency at analysis time. The more factors you have with more than two or three levels, the harder it is usually going to be to find an orthogonal and balanced design or even a close approximation.

There are times, however, when you can reasonably create factors with more levels than you really need. Say you have a design with two-level and four-level factors and you want to create quadratic price effects, which would require three evenly-spaced levels. Say you also want the ability to test for departures from a quadratic model. You could use a strategy that begins with an eight-level price factor. Then you can recode it as follows: (1 2 3 4 5 6 7 8) $\rightarrow$ (1 2 3 4 5 1 3 5). Notice that you end up with twice as many points at the minimum, middle, and maximum positions as in the second and fourth positions. This gives you good efficiency for the quadratic effect and some information about higher-order effects. Furthermore, there are many designs with mixtures of (2, 4, and 8)-level factors in 64 runs, which you can easily block. In contrast, you need 400 runs ($4 \times 4 \times 5 \times 5$) before you can find a design such as $2^2 4^2 5^2$ in an orthogonal array. If you are assigning levels with a format, you can assign levels and do the recoding all at the same time as in the following example:

```
proc format;
   value price  1 = $2.89 2 = $2.99 3 = $3.09 4 = $3.19 5 = $3.29
                6 = $2.89           7 = $3.09           8 = $3.29;
   run;
```

Randomization

Randomization is the process of sorting the rows of a design into a random order and randomly reassigning all of the factor levels. See page 57 for an example of randomization. Some designs must be randomized before you use them. Full-factorial designs, fraction-factorial designs, and orthogonal arrays all need to be randomized. These designs must be randomized for subject-effect reasons not for statistical reasons. Randomization does not change a design's efficiency, orthogonality, or balance. Hence, statistically, it does not matter. The problem with many full-factorial designs, fraction-factorial designs, and orthogonal arrays is they have some rows have a recognizable pattern of levels. Most typically, there are two problems with the original design. Often, the rows are sorted (or the levels change in some other predictable way), and often, one row is constant. Typically, when a row is constant, the first row consists entirely of the first factor level. The `%MktEx` macro automatically randomizes designs and stores the randomized design in the `outr=` data set.

Designs created through a coordinate exchange or candidate set search do not need to be randomized, since the construction algorithm has a random component. However, randomizing these designs is fine as long as there are no restrictions on the design. Randomization occurs after the design is constructed with the restrictions imposed. If you restrict the design, say by preventing a certain level of one factor from appearing with a certain level of another factor, and then randomize, the restriction will quite likely be violated in the randomized design.

Typically, designs for linear models, such as those used for conjoint models, are randomized. Linear model designs that are later used to make choice designs are also typically randomized. However, once a design is in choice-design form, it should not be randomized. For example, exchanging rows could in many cases destroy the integrity of the design. At best, it could decrease the efficiency of the choice design. Randomization is typically not required for choice designs because often, some randomization has occurred earlier in their construction.

Random Number Seeds

Our designs are created in many ways, but virtually every design we create uses a random number stream as part of the process. In some cases, the initial design is random, then a computerized search is used to improve it. In other cases, an orthogonal array is used, but first it is randomized (the rows are sorted into a random order and the factor levels are randomly reassigned). There are many other ways in which random number streams are used in making designs. All of the SAS design macros that have a random component have a `seed=` option to control the random number seed. The random number seed is an integer in the range 1 to 2,147,483,646. This seed is used to provide a starting point for the random number stream. Note, however, that the seed itself is not a part of the stream. If you do not specify a seed, then a seed is generated for you and displayed in the SAS log. The default seed is a (nonobvious) function of the date and the time. Most of the examples in this book use an explicitly set random number seed. The few exceptions are cases where orthogonal arrays or full-factorial designs (without randomization) are generated, and the seed does not matter.

Explicitly specifying the random number seed is a good programming practice. If you do not specify a seed, and you are creating a design with a random component, then you probably will get different results each time you run the macro. You will certainly get different results unless your design is very small. Even with only a single two-level factor, you will only get the same results 50% of the time. With different seeds, you should expect efficiencies that are similar, but you should not expect them to always be the same.

It is important to not only specify the seed, but it is also important that you save your design for later use in the analysis. If by chance you make a design and then install new macros, get a new computer, or update the operating system or SAS release, even knowing the seed might not be enough to reproduce the same design. Algorithms with random components are not guaranteed to always produce the same results when things change. Imagine an ant climbing a large sand dune, each time stepping only on higher grains of sand. Then imagine a light breeze slightly shifting the sand. A second ant will quite likely find a slightly different path even if she starts from the same place as the first ant.

Sometimes, when you run one of the design macros, you might get results that are undesirable for some reason. Perhaps there are duplicates but you did not specify `options=nodups`. Perhaps one factor is slightly less balanced than you would like. There are many other things that could happen. Sometimes changing the random number seed is enough to make the undesirable results go away. Other times you need to take more aggressive approaches such as specifying new options.

Duplicates

It is sometimes the case that an optimal experimental design will have duplicates. For a linear arrangement or conjoint design, this means duplicate runs or profiles. In a choice design, this can mean duplicate choice sets or duplicate profiles within a choice set. Consider the following orthogonal array in 12 runs:

```
1  1  1  2  1  1  2  1  2  2  2
1  1  2  1  1  2  1  2  2  2  1
1  1  2  1  2  2  2  1  1  1  2
1  2  1  1  2  1  2  2  2  1  1
1  2  1  2  2  2  1  1  1  2  1
1  2  2  2  1  1  1  2  1  1  2
2  1  1  1  2  1  1  2  1  2  2
2  1  1  2  1  2  2  2  1  1  1
2  1  2  2  2  1  1  1  2  1  1
2  2  1  1  1  2  1  1  2  1  2
2  2  2  1  1  1  2  1  1  2  1
2  2  2  2  2  2  2  2  2  2  2
```

This is an optimal design with 11 factors and with no duplicates. Any subset of columns is also an optimal design but with a reduced number of factors. However, if you select only the first four columns, then the second and third rows are duplicates. Often, researchers will want to avoid designs with duplicates. The reasons will typically have more to do with worries about what the subjects will think if given the same task twice or what the client or brand manager will think. That is, the reasons to worry about duplicates are based on human concerns not statistical concerns. Duplicates do not pose any problem from a statistical or design efficiency point of view. In fact, the maximum D-efficiency for the design 2^4 in 12 runs with duplicates is 100 and without duplicates it is 97.6719. Still, human concerns *are* important in making a design, so often researchers try to avoid duplicates.

You can use the `%MktDups` macro to check your linear or choice design for duplicates. In practice, duplicates do not occur very often. When they occur, sometimes something as simple as changing the random number seed is sufficient to avoid duplicates. When that fails, you can ensure that duplicates are not created with macro options. You can specify `options=nodups` with both the `%MktEx` and `%ChoicEff` macros. The `nodups` specification is not the default because it imposes time-consuming restrictions on the algorithm that are rarely needed. See the documentation for the `%MktDups` macro beginning on page 1004 for more information about duplicates. In addition, there are numerous examples of using this macro throughout this book including in this chapter on the following pages: 147, 174, 198, and 206.

Orthogonal Arrays and Difference Schemes

This section provides some details about how certain orthogonal arrays are constructed. This section also discusses difference schemes which are fundamental building blocks in orthogonal array and optimal generic choice design construction. This section is optional and can be skipped by all but the most interested of readers. The next section starts on page 101. This section begins by illustrating how the first (that is, smallest) five orthogonal arrays are constructed. They were previously shown on page 59. These designs require the easiest of orthogonal array construction methods. The point of this discussion is not to fully explain how orthogonal arrays are constructed or even fully explain how these orthogonal arrays are constructed. Rather, the goal is just to provide the tiniest glimpse into the methods that the `%MktEx` macro uses to construct orthogonal arrays, to illustrate some of their beauty, and to show some of the ways they are connected.

The five designs that were previously displayed on page 59 are now displayed with a different set of integers representing the levels. The following representation is more natural from a design construction point of view:

2^3		
1	1	1
-1	1	-1
1	-1	-1
-1	-1	1

$2^1 3^1$	
0	0
0	1
0	2
1	0
1	1
1	2

2^7						
1	1	1	1	1	1	1
-1	1	-1	1	-1	1	-1
1	-1	-1	1	1	-1	-1
-1	-1	1	1	-1	-1	1
1	1	1	-1	-1	-1	-1
-1	1	-1	-1	1	-1	1
1	-1	-1	-1	-1	1	1
-1	-1	1	-1	1	1	-1

$2^4 4^1$				
1	1	1	1	0
1	-1	1	-1	1
1	1	-1	-1	2
1	-1	-1	1	3
-1	-1	-1	-1	0
-1	1	-1	1	1
-1	-1	1	1	2
-1	1	1	-1	3

3^4			
0	0	0	0
0	1	2	1
0	2	1	2
1	1	1	0
1	2	0	1
1	0	2	2
2	2	2	0
2	0	1	1
2	1	0	2

We will consider the second design first. It is a full-factorial design, and hence it is trivially constructed by making all combinations of the levels.

Next, consider the following matrix:

$$\mathbf{H}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

This matrix is called a Hadamard matrix of order 2. In an experiment with two treatments, the first column represents the intercept or grand mean, and the second column represents a contrast between the two group means. Next, consider the Kronecker product of $\mathbf{H}_2$ with itself:

$$\mathbf{H}_4 = \mathbf{H}_2 \otimes \mathbf{H}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} h_{11}\mathbf{H}_2 & h_{12}\mathbf{H}_2 \\ h_{21}\mathbf{H}_2 & h_{22}\mathbf{H}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{H}_2 & \mathbf{H}_2 \\ \mathbf{H}_2 & -\mathbf{H}_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

The last three columns of $\mathbf{H}_4$ form the design 2^3 in four runs and the first column provides an intercept. $\mathbf{H}_4$ also matches the first four rows of the two-level factors in $2^4 4^1$ in eight runs, and $-\mathbf{H}_4$ matches the last four rows. The last column of $2^4 4^1$ consists of two repetitions of (0, 1, 2, 3).

Next, consider the Kronecker product of $\mathbf{H}_2$ with $\mathbf{H}_4$:

$$\mathbf{H}_8 = \mathbf{H}_2 \otimes \mathbf{H}_4 = \begin{bmatrix} \mathbf{H}_4 & \mathbf{H}_4 \\ \mathbf{H}_4 & -\mathbf{H}_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}$$

The last seven columns of $\mathbf{H}_8$ form the design 2^7 in eight runs and the first column provides an intercept.

Next, consider the following matrix product:

$$\mathbf{D}_3 = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} \begin{bmatrix} 0 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 2 & 1 \end{bmatrix}$$

Note that arithmetic in this case is field arithmetic, not real number arithmetic, so all results are mod 3. More is said about this later in this section. In this case, the only result different from real number arithmetic is $2 \times 2 = 1$ (which comes from 2×2 mod $3 = 4$ mod $3 = 1$, where n mod m returns the remainder after dividing n by m). $\mathbf{D}_3$ matches the first three rows and columns of 3^4 in nine runs. The full first three columns are as follows:

$$\begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} \oplus \mathbf{D}_3 = \begin{bmatrix} 0 + \mathbf{D}_3 \\ 1 + \mathbf{D}_3 \\ 2 + \mathbf{D}_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 2 & 1 \\ 1 & 1 & 1 \\ 1 & 2 & 0 \\ 1 & 0 & 2 \\ 2 & 2 & 2 \\ 2 & 0 & 1 \\ 2 & 1 & 0 \end{bmatrix}$$

The $\oplus$ operator denotes an operation like the Kronecker product but elements are added rather than multiplied. Again, arithmetic is mod 3 (hence, $2 + 1 = 0$, $2 + 2 = 1$). The last column of 3^4 consists of three repetitions of (0, 1, 2). The matrix $\mathbf{D}_3$ is called a *difference scheme*, and this construction method is called "developing a difference scheme."

An orthogonal array $p^1 m^q$ in $p \times m$ runs is made by developing a difference scheme (Wangand Wu 1991). A difference scheme is a matrix that is a "building block" used in the construction of many orthogonal arrays. It is called a difference scheme because if you subtract any two columns, all differences occur equally often. Note that like field addition, subtraction in a field is quite different from subtraction in the real number system. Here, arithmetic operations are in a Galois or abelian field. Explaining this fully is beyond the scope of this discussion, but we will provide an example. Specifically, we consider the case where $p = m = q = 5$. The following tables show the addition, subtraction, multiplication, and inversion tables that are used in a Galois field of order 5 (GF(5)):

Addition

	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Subtraction

	0	1	2	3	4
0	0	4	3	2	1
1	1	0	4	3	2
2	2	1	0	4	3
3	3	2	1	0	4
4	4	3	2	1	0

Multiplication

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Inverse

0	
1	1
2	3
3	2
4	4

These tables are used when constructing factors with five levels (0 1 2 3 4). In this case, since the order, 5, is a prime number, the rules for addition and multiplication follow the rules for integer arithmetic mod 5. For example, $4 + 4 \bmod 5 = 8 \bmod 5 = 3$ and $4 \times 4 \bmod 5 = 16 \bmod 5 = 1$. These results can also be seen by accessing the row 4, column 4 entries of the addition and multiplication tables. The rules for subtraction can easily be derived from the rules for addition, and the rules for inversion can easily be derived from the rules for multiplication. For example, since $4 + 3 = 2$ in GF(5), then $4 = 2 - 3$, and since $3 \times 2 = 1$, then 2 is the inverse of 3. Note that in many cases, the rules for field arithmetic are not this simple. In some cases, such as when the order of the field is a power of a prime (4, 8, 9 ...) or a composite number that contains a power of a prime (12, 18, ...), the rules are much more complicated, and modulo m arithmetic does not work.

Let $\ell_5' = [0\ 1\ 2\ 3\ 4]$ be a row vector with the field elements and $\mathbf{1}_5$ be a column vector with 5 ones. In GF(5), the multiplication table is a 5×5 difference scheme, $\mathbf{D}_5 = \ell_5 \ell_5'$ (where $\ell_5 \ell_5'$ arithmetic, of course, occurs in GF(5)). You can verify that if you subtract every column from every other column, the five elements in ℓ_5 all occur exactly once in all of the difference vectors. The following matrix is the orthogonal array 5^6 in 25 runs:

$$\left[\begin{array}{cc} \mathbf{1}_5 \otimes \ell_5 & \ell_5 \oplus \mathbf{D}_5 \end{array}\right] = \left[\begin{array}{cc} \ell_5 & 0 + \mathbf{D}_5 \\ \ell_5 & 1 + \mathbf{D}_5 \\ \ell_5 & 2 + \mathbf{D}_5 \\ \ell_5 & 3 + \mathbf{D}_5 \\ \ell_5 & 4 + \mathbf{D}_5 \end{array}\right]$$

This matrix is partitioned vertically into five blocks of five rows and horizontally into a column followed by a set of five columns. For each new row block, the difference scheme is shifted by adding 1 to the previous matrix. The resulting orthogonal array is shown in Table 15 on the left, and the generic choice design made by sorting this orthogonal array on the first factor is shown on the right.

This is the same orthogonal array that `%MktEx` produces except that by default, `%MktEx` uses one-based integers instead of a zero base. For a generic choice design, the difference scheme provides levels for the first alternative, and all other alternatives are made from the previous alternative by adding 1 in the appropriate field.

You can use the `%MktEx` macro to make difference schemes by selecting just the right rows and columns of a design. The following steps create and display a 6×6 difference scheme of order 3:

```
%mktex(3 ** 6 6,                    /* factors, difference scheme in 3 ** 6 */
       n=18,                        /* number of runs in full design       */
       options=nosort,              /* do not sort the design              */
       levels=0)                    /* make levels 0, 1, 2 (not 1, 2, 3)    */

proc print data=design(obs=6) noobs; var x1-x6; run;
```

Table 15

Orthogonal Array

0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1
0	1	1	1	1	1
1	1	2	3	4	0
2	1	3	0	2	4
3	1	4	2	0	3
4	1	0	4	3	2
0	2	2	2	2	2
1	2	3	4	0	1
2	2	4	1	3	0
3	2	0	3	1	4
4	2	1	0	4	3
0	3	3	3	3	3
1	3	4	0	1	2
2	3	0	2	4	1
3	3	1	4	2	0
4	3	2	1	0	4
0	4	4	4	4	4
1	4	0	1	2	3
2	4	1	3	0	2
3	4	2	0	3	1
4	4	3	2	1	0

Generic Choice Design

Set	Attributes				
0	0	0	0	0	0
0	1	1	1	1	1
0	2	2	2	2	2
0	3	3	3	3	3
0	4	4	4	4	4
1	0	1	2	3	4
1	1	2	3	4	0
1	2	3	4	0	1
1	3	4	0	1	2
1	4	0	1	2	3
2	0	2	4	1	3
2	1	3	0	2	4
2	2	4	1	3	0
2	3	0	2	4	1
2	4	1	3	0	2
3	0	3	1	4	2
3	1	4	2	0	3
3	2	0	3	1	4
3	3	1	4	2	0
3	4	2	0	3	1
4	0	4	3	2	1
4	1	0	4	3	2
4	2	1	0	4	3
4	3	2	1	0	4
4	4	3	2	1	0

The difference scheme is as follows:

```
x1    x2    x3    x4    x5    x6

 0     0     0     0     0     0
 0     0     1     1     2     2
 0     1     0     2     2     1
 0     1     2     0     1     2
 0     2     1     2     1     0
 0     2     2     1     0     1
```

The difference scheme is the matrix $\mathbf{D}_6$ from the orthogonal array $3^6 6^1$ in 18 runs which is created by `%MktEx` as follows:

$$\left[\begin{array}{cc} \ell_3 \oplus \mathbf{D}_6 & \mathbf{1}_3 \otimes \ell_6 \end{array}\right] = \left[\begin{array}{cc} 0 + \mathbf{D}_6 & \ell_6 \\ 1 + \mathbf{D}_6 & \ell_6 \\ 2 + \mathbf{D}_6 & \ell_6 \end{array}\right]$$

While it is easy to make an $m \times m$ difference scheme of order m when m is prime using mod m arithmetic, it is hard to make a difference scheme of order $2m \times 2m$ order m and most larger difference schemes without software such as the `%MktEx` macro.

You can easily verify that this difference scheme, using each row (plus one) as a first alternative, and cyclic shifting for the other alternatives, makes the optimal choice design on page 109. You could construct the same design directly from the difference scheme as follows:

```
data choice(keep=set x1-x6);
   Set = _n_;
   set design(obs=6);
   array x[6];
   do i = 1 to 6; x[i] + 1; end;
   output;
   do i = 1 to 6; x[i] = mod(x[i], 3) + 1; end;
   output;
   do i = 1 to 6; x[i] = mod(x[i], 3) + 1; end;
   output;
   run;

proc print; id set; by set; run;
```

The design is follows:

```
Set    x1    x2    x3    x4    x5    x6

 1      1     1     1     1     1     1
        2     2     2     2     2     2
        3     3     3     3     3     3

 2      1     1     2     2     3     3
        2     2     3     3     1     1
        3     3     1     1     2     2

 3      1     2     1     3     3     2
        2     3     2     1     1     3
        3     1     3     2     2     1

 4      1     2     3     1     2     3
        2     3     1     2     3     1
        3     1     2     3     1     2
```

```
5      1    3    2    3    2    1
       2    1    3    1    3    2
       3    2    1    2    1    3

6      1    3    3    2    1    2
       2    1    1    3    2    3
       3    2    2    1    3    1
```

Canonical Correlations

We use canonical correlations to evaluate nonorthogonal designs and the extent to which factors are correlated or are not independent. To illustrate, consider the following design with four three-level factors in 9 runs:

Linear Arrangement

x1	x2	x3	x4
1	1	1	1
1	2	3	3
1	3	2	2
2	1	2	3
2	2	1	2
2	3	3	1
3	1	3	2
3	2	2	1
3	3	1	3

Coded Linear Arrangement

x1			x2			x3			x4		
1	2	3	1	2	3	1	2	3	1	2	3
1	0	0	1	0	0	1	0	0	1	0	0
1	0	0	0	1	0	0	0	1	0	0	1
1	0	0	0	0	1	0	1	0	0	1	0
0	1	0	1	0	0	0	1	0	0	0	1
0	1	0	0	1	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0	1	1	0	0
0	0	1	1	0	0	0	0	1	0	1	0
0	0	1	0	1	0	0	1	0	1	0	0
0	0	1	0	0	1	1	0	0	0	0	1

Each three-level factor can be coded with three columns that contain the less-than-full-rank binary coding (see page 73). A factor can be recoded by applying a coefficient vector $\boldsymbol{\alpha}' = (\alpha_1 \; \alpha_2 \; \alpha_3)$ or $\boldsymbol{\beta}' = (\beta_1 \; \beta_2 \; \beta_3)$ to a coded factor to create a single column. In other words, the original coding of (1 2 3) can be replaced with arbitrary $(\alpha_1 \; \alpha_2 \; \alpha_3)$ or $(\beta_1 \; \beta_2 \; \beta_3)$. If two factors are orthogonal, then for all choices of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, the simple correlation between recoded columns is zero. A *canonical correlation* shows the maximum correlation between two recoded factors that can be obtained with the optimal $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. This design, 3^4 in 9 runs is orthogonal so for all pairs of factors and all choices of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, the simple correlation between recoded factors is zero. The canonical correlation between a factor and itself is 1.0.

For nonorthogonal designs and designs with interactions, the canonical-correlation matrix is not a substitute for looking at the variance matrix discussed on pages 351, 425, and 1058. It just provides a quick and more-compact picture of the correlations between the factors. The variance matrix is sensitive to the actual model specified and the actual coding. The canonical-correlation matrix just tells you if there is some correlation between the main effects. A matrix of canonical correlations provides a useful picture of the orthogonality or lack of orthogonality in a design. For example, the following canonical-correlation matrix from the vacation example on page 350, shows a design with 16 factors that are mostly orthogonal:

Canonical Correlation Matrix

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16
x1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
x4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
x5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
x6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
x7	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
x8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
x9	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
x10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
x11	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
x12	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
x13	0	0	0	0	0	0	0	0	0	0	0	0	1	0.25	0.25	0
x14	0	0	0	0	0	0	0	0	0	0	0	0	0.25	1	0.25	0
x15	0	0	0	0	0	0	0	0	0	0	0	0	0.25	0.25	1	0
x16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

However, `x13-x15` are not orthogonal to each other. Still, with $r^2 = 0.25^2 = 0.0625$, these factors are nearly independent.

Optimal Generic Choice Designs

In some situations, particularly for certain generic choice experiments, we can make optimal choice designs under the assumption that $\boldsymbol{\beta} = 0$. The idea of optimal generic choice designs started with the work of Don Anderson (Bunch, Louviere, and Anderson 1996) who introduced the idea of creating these designs by shifting alternatives in orthogonal arrays. We approach optimal generic designs only slightly differently—from the point of view of difference scheme development and orthogonal array selection. The approach I discuss here grew from numerous discussions that I have had with Don Anderson over the years. Optimal generic choice designs are discussed extensively (much more extensively than here) by Street and Burgess (2007) who provide a great deal of theory.

Our goal in this section is to construct optimal generic designs and get a report of the design's efficiency on a 0 to 100 scale like we get with linear model designs. A generic choice experiment is one that does not have any brands. The alternatives are simply bundles of attributes. For example, a manufacturer of any electronic product might construct a choice study with potential variations on a new product to see which attributes are the most important drivers of choice. Consider a study that involves 4 two-level factors and four choice sets, each with two alternatives. The following tables display an the optimal generic choice design and show how it is constructed:

Optimal Generic Choice Design

1	1	1	1
2	2	2	2
1	1	2	2
2	2	1	1
1	2	2	1
2	1	1	2
1	2	1	2
2	1	2	1

Fractional Factorial 2^3 in 4 Runs With Intercept

1	1	1	1
1	1	2	2
1	2	2	1
1	2	1	2

Shifted Fractional Factorial

2	2	2	2
2	2	1	1
2	1	1	2
2	1	2	1

Orthogonal Array $4^1 2^4$ in 8 Runs

1	1	1	1	1
2	1	1	2	2
3	1	2	2	1
4	1	2	1	2
1	2	2	2	2
2	2	2	1	1
3	2	1	1	2
4	2	1	2	1

Sorted Orthogonal Array

1	1	1	1	1
1	2	2	2	2
2	1	1	2	2
2	2	2	1	1
3	1	2	2	1
3	2	1	1	2
4	1	2	1	2
4	2	1	2	1

The fractional-factorial design consists of 3 two-level factors in 4 runs and an intercept (the customary column of ones). The shifted design consists of 3 two-level factors in 4 runs along with a column of twos (the intercept plus 1) instead of the customary column of ones. The final table is a mixed orthogonal array with 1 four-level factor and 4 two-level factors in eight runs. Using the notation discussed in the section beginning on page 95, this design is constructed by adding 1 to the following orthogonal array:

$$\left[\begin{array}{cc} \mathbf{1}_2 \otimes \ell_4 & \ell_2 \oplus \mathbf{D}_4 \end{array}\right] = \left[\begin{array}{cc} \ell_4 & 0 + \mathbf{D}_4 \\ \ell_4 & 1 + \mathbf{D}_4 \end{array}\right]$$

The first fractional-factorial design exactly matches the two-level factors in the first half of the third fractional-factorial design, and the second table exactly matches the two-level factors in the second half of the fractional-factorial design in the third table. Sorting this design on the four-level factor and using the four-level factor as the choice set number yields the optimal generic choice design.

The optimal generic choice design is constructed by creating a fractional-factorial design with an intercept and using it to make the first alternative of each choice set. The second alternative is made from the first by shifting or cycling through the levels (changing 1 to 2 and 2 to 1). The first alternative is shown in the fractional-factorial table, and the second alternative is shown in shifted fractional-factorial table. The plan for the second alternative is a different fractional-factorial plan. Alternatively, equivalently, and more clearly, this design can be made from the orthogonal array $4^1 2^4$ in 8 runs by using the four-level factor as the choice set number. Note that the optimal generic choice design never shows two alternatives with the same levels of any factor. For this reason, some researchers do not use them and consider this class of designs to be more of academic and combinatorial interest than of practical significance.

A randomized version of this design (where the first choice set will not consist of constant attributes within each alternative) is constructed and evaluated as follows:

```
%mktex(4 2 ** 4,                  /* choice set number and attr levels    */
       n=8)                       /* 8 runs - 4 sets, two alts each       */

%mktlab(data=randomized,          /* randomized design                    */
        vars=Set x1-x4)           /* var names for set var and for attrs  */

proc sort; by set; run;

proc print; by set; id set; run;

%choiceff(data=final,             /* candidate set of choice sets         */
          init=final(keep=set),   /* select these sets from candidates    */
          model=class(x1-x4 / sta), /* model with stdzd orthogonal coding  */
          nsets=4,                /* 4 choice sets                        */
          nalts=2,                /* 2 alternatives per set               */
          options=relative,       /* display relative D-efficiency        */
          beta=zero)              /* assumed beta vector, Ho: b=0         */
```

The following boxes show an optimal generic choice design with 9 three-level attributes, with three alternatives, and nine choice sets, each in a separate box:

1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3

1	2	3	3	1	2	2	3	1
2	3	1	1	2	3	3	1	2
3	1	2	2	3	1	1	2	3

1	1	1	2	2	2	3	3	3
2	2	2	3	3	3	1	1	1
3	3	3	1	1	1	2	2	2

1	3	2	1	3	2	1	3	2
2	1	3	2	1	3	2	1	3
3	2	1	3	2	1	3	2	1

1	1	1	3	3	3	2	2	2
2	2	2	1	1	1	3	3	3
3	3	3	2	2	2	1	1	1

1	3	2	2	1	3	3	2	1
2	1	3	3	2	1	1	3	2
3	2	1	1	3	2	2	1	3

1	2	3	1	2	3	1	2	3
2	3	1	2	3	1	2	3	1
3	1	2	3	1	2	3	1	2

1	3	2	3	2	1	2	1	3
2	1	3	1	3	2	3	2	1
3	2	1	2	1	3	1	3	2

1	2	3	2	3	1	3	1	2
2	3	1	3	1	2	1	2	3
3	1	2	1	2	3	2	3	1

It is made from the orthogonal design $3^9 9^1$ in 27 runs by using the nine-level factor as the choice set number. Notice that each alternative is made from the previous alternative by adding one to the previous level, mod 3.[†] Similarly, the first alternative is made from the third alternative by adding one to the previous level, mod 3. A randomized version of this design (where the first choice set will not consist of constant attributes within each alternative) is constructed and evaluated as follows:

[†]More precisely, since these numbers are based on one instead of zero, the operation is: (x mod 3) + 1.

```
%mktex(9 3 ** 9,                   /* choice set number and attr levels    */
       n=27)                       /* 27 runs - 9 sets, 3 alts each        */

%mktlab(data=randomized,           /* randomized design                    */
        vars=Set x1-x9)            /* var names for set var and for attrs  */

proc sort; by set; run;

proc print; by set; id set; run;

%choiceff(data=final,              /* candidate set of choice sets         */
          init=final(keep=set),    /* select these sets from candidates    */
          model=class(x1-x9 / sta), /* model with stdzd orthogonal coding  */
          nsets=9,                 /* 9 choice sets                        */
          nalts=3,                 /* 3 alternatives per set               */
          options=relative,        /* display relative D-efficiency        */
          beta=zero)               /* assumed beta vector, Ho: b=0         */
```

An optimal generic choice design with 8 four-level attributes, with four alternatives, and eight choice sets, each in a separate box is shown as follows:

1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4

1	3	2	1	4	2	3	4
2	4	1	2	3	1	4	3
3	1	4	3	2	4	1	2
4	2	3	4	1	3	2	1

1	1	3	4	2	2	4	3
2	2	4	3	1	1	3	4
3	3	1	2	4	4	2	1
4	4	2	1	3	3	1	2

1	3	4	4	3	1	2	2
2	4	3	3	4	2	1	1
3	1	2	2	1	3	4	4
4	2	1	1	2	4	3	3

1	2	2	3	3	4	4	1
2	1	1	4	4	3	3	2
3	4	4	1	1	2	2	3
4	3	3	2	2	1	1	4

1	4	1	3	2	3	2	4
2	3	2	4	1	4	1	3
3	2	3	1	4	1	4	2
4	1	4	2	3	2	3	1

1	2	4	2	4	3	1	3
2	1	3	1	3	4	2	4
3	4	2	4	2	1	3	1
4	3	1	3	1	2	4	2

1	4	3	2	1	4	3	2
2	3	4	1	2	3	4	1
3	2	1	4	3	2	1	4
4	1	2	3	4	1	2	3

It is made from the fractional-factorial design $4^8 8$ in 32 runs by using the eight-level factor as the choice set number. Notice that every attribute has all four levels in each factor. With four-level factors, the rules that are used to make orthogonal arrays are more complicated than the mod 3 addition that is used with three-level factors, so you do not get the same pattern of shifted results that we saw previously. A randomized version of this design (where the first choice set will not consist of constant attributes within each alternative) is constructed and evaluated as follows:

```
%mktex(8 4 ** 8,                  /* choice set number and attr levels    */
       n=32)                      /* 32 runs - 8 sets, four alts each     */

%mktlab(data=randomized,          /* randomized design                    */
        vars=Set x1-x8)           /* var names for set var and for attrs  */

proc sort; by set; run;

proc print; by set; id set; run;

%choiceff(data=final,             /* candidate set of choice sets         */
          init=final(keep=set),   /* select these sets from candidates    */
          model=class(x1-x8 / sta), /* model with stdzd orthogonal coding   */
          nsets=8,                /* 8 choice sets                        */
          nalts=4,                /* 4 alternatives per set               */
          options=relative,       /* display relative D-efficiency        */
          beta=zero)              /* assumed beta vector, Ho: b=0         */
```

If you need a generic choice design and you do not have the level of symmetry shown in these examples (all m-level factors with m alternatives) then you can use the `%ChoicEff` macro to find an efficient generic design using the methods shown on page 198 and in the chair example on page 556. Also see the documentation for the `%ChoicEff` macro beginning on page 806 for examples of generic design construction.

An interesting class of optimal generic designs can be constructed for experiments with p choice sets and m-level factors with m alternatives when there is an orthogonal array $p^1 m^q$ in $p \times m$ runs where $q \leq p$. We can process the design catalog from the `%MktOrth` macro to find these as follows:

```
%mktorth(maxn=100,                /* output up to 100 runs                */
         options=parent)          /* just list the parent designs         */

data x;
   set mktdeslev;
   array x[50];
   gotone = 0;
   do p = 50 to 1 by -1 until(gotone);
      if x[p] eq 1 then gotone = 1;
      end;
   if not gotone then do;
      p = sqrt(n);
      if abs(p * p - n) < 1e-8 then if x[p] then gotone = 1;
      end;
      if gotone then do;
         m = n / p;
         if m eq p then x[m] + -1;
         q = x[m];
         design = compbl(put(m, 5.) || ' ** ' || put(q, 5.));
         From   = compbl(put(p, 5.) || ' ** 1 ' ||
                         trim(design) || ', n=' || put(n, 5. -L));
         if (n le 10 and q > 1) or (n gt 10 and q > 2) then output;
         end;
      run;

   proc sort;
      by n p descending q;
      run;

   data list(keep=sets alts design from);
      length Sets Alts 8;
      set x;
      by n p;
      if first.p;
      sets = p;
      alts = n / p;
      run;

   proc sort;
      by sets descending alts;
      run;

   proc print; by sets; id sets; run;
```

The final `if` statement in the first DATA step filters out designs that technically meet this definition, but are mostly uninteresting. Specifically, a number of designs have exactly 2 two-level factors in varying numbers of choice sets. These are filtered out by the last clause of the `if` statement. A few of the smaller designs that work are as follows:

```
Sets    Alts    Design      From

  2       2     2 ** 2      2 ** 1 2 ** 2, n=4

  3       3     3 ** 3      3 ** 1 3 ** 3, n=9

  4       4     4 ** 4      4 ** 1 4 ** 4, n=16
          2     2 ** 4      4 ** 1 2 ** 4, n=8

  5       5     5 ** 5      5 ** 1 5 ** 5, n=25

  6       3     3 ** 6      6 ** 1 3 ** 6, n=18

  7       7     7 ** 7      7 ** 1 7 ** 7, n=49
  8       8     8 ** 8     8 ** 1 8 ** 8, n=64
          4     4 ** 8     8 ** 1 4 ** 8, n=32
          2     2 ** 8     8 ** 1 2 ** 8, n=16

  9       9     9 ** 9     9 ** 1 9 ** 9, n=81
          3     3 ** 9     9 ** 1 3 ** 9, n=27

 10      10     10 ** 3    10 ** 1 10 ** 3, n=100
          5     5 ** 10    10 ** 1 5 ** 10, n=50

 12       6     6 ** 6     12 ** 1 6 ** 6, n=72
          4     4 ** 12    12 ** 1 4 ** 12, n=48
          3     3 ** 12    12 ** 1 3 ** 12, n=36
          2     2 ** 12    12 ** 1 2 ** 12, n=24

 14       7     7 ** 14    14 ** 1 7 ** 14, n=98

 15       5     5 ** 8     15 ** 1 5 ** 8, n=75
          3     3 ** 9     15 ** 1 3 ** 9, n=45

 16       4     4 ** 16    16 ** 1 4 ** 16, n=64
          2     2 ** 16    16 ** 1 2 ** 16, n=32

 18       3     3 ** 18    18 ** 1 3 ** 18, n=54

 20       5     5 ** 20    20 ** 1 5 ** 20, n=100
          4     4 ** 10    20 ** 1 4 ** 10, n=80
          2     2 ** 20    20 ** 1 2 ** 20, n=40

 21       3     3 ** 12    21 ** 1 3 ** 12, n=63

 24       4     4 ** 20    24 ** 1 4 ** 20, n=96
          3     3 ** 24    24 ** 1 3 ** 24, n=72
          2     2 ** 24    24 ** 1 2 ** 24, n=48

 27       3     3 ** 27    27 ** 1 3 ** 27, n=81

 28       2     2 ** 28    28 ** 1 2 ** 28, n=56

 30       3     3 ** 30    30 ** 1 3 ** 30, n=90

 32       2     2 ** 32    32 ** 1 2 ** 32, n=64

 33       3     3 ** 13    33 ** 1 3 ** 13, n=99

 36       2     2 ** 36    36 ** 1 2 ** 36, n=72

 40       2     2 ** 40    40 ** 1 2 ** 40, n=80

 44       2     2 ** 44    44 ** 1 2 ** 44, n=88
```

```
                48        2     2 ** 48    48 ** 1 2 ** 48, n=96
```

It is important to note that this is not the complete list of small orthogonal arrays that can be developed into optimal generic choice designs. Rather these are the most interesting ones that work in the symmetric case, that is, where all levels of all factors are the same. More is said about this later in this section.

For a specification of p, q, and m, assuming the orthogonal array $p^1 m^q$ in pm runs exists, the following steps make an optimal generic choice design and use the `%ChoicEff` macro to evaluate the results:

```
%let p = 6;                        /* p - number of choice sets           */
%let m = 3;                        /* m-level factors                     */
%let q = &p;                       /* q - number of factors               */

%mktex(&p &m ** &q,                /* choice set number and attr levels   */
       n=&p * &m)                  /* p * m runs - p sets, m alts each    */

%mktlab(data=design,               /* orthogonal array                    */
        vars=Set x1-x&q)           /* var names for set var and for attrs */

proc print; id set; by set; run;

%choiceff(data=final,              /* candidate set of choice sets        */
          init=final(keep=set),    /* select these sets from candidates   */
          model=class(x1-x&q / sta),/* model with stdzd orthogonal coding  */
          nsets=&p,                /* &p choice sets                      */
          nalts=&m,                /* &m alternatives per set             */
          options=relative,        /* display relative D-efficiency       */
          beta=zero)               /* assumed beta vector, Ho: b=0        */
```

The `sta` (short for `standorth`) option in the `model=` option is new with SAS 9.2 and requests a standardized orthogonal contrast coding. You must specify this coding if you want to see relative D-efficiency on a 0 to 100 scale. Relative D-efficiency is not displayed by default since it is only meaningful for a limited class of designs such as these designs. You must request it with `options=relative`. Relative D-efficiency is explained in more detail in starting on page 81 and in this example as more results are presented. The following example has $m = 3$ and $p = q = 6$ choice sets:

```
                           Set    x1    x2    x3    x4    x5    x6

                            1      1     1     1     1     1     1
                                   2     2     2     2     2     2
                                   3     3     3     3     3     3

                            2      1     1     2     2     3     3
                                   2     2     3     3     1     1
                                   3     3     1     1     2     2
```

```
    3       1     2     1     3     3     2
            2     3     2     1     1     3
            3     1     3     2     2     1

    4       1     2     3     1     2     3
            2     3     1     2     3     1
            3     1     2     3     1     2

    5       1     3     2     3     2     1
            2     1     3     1     3     2
            3     2     1     2     1     3

    6       1     3     3     2     1     2
            2     1     1     3     2     3
            3     2     2     1     3     1
```

The results summary table that follows the iteration history is as follows:

```
                     Final Results

               Design                    1
               Choice Sets               6
               Alternatives              3
               Parameters               12
               Maximum Parameters       12
               D-Efficiency         6.0000
               Relative D-Eff     100.0000
               D-Error              0.1667
               1 / Choice Sets      0.1667
```

This table shows that *D*-Efficiency is 6. *D*-Error is $1/6 \approx 0.1667$, since *D*-Error is always the inverse of the *D*-Efficiency. It shows that a design with a relative *D*-efficiency of 100 was found. It also shows that this design is saturated—the number of parameters and the maximum number of parameters are the same. The parameters names and their variances under the null hypothesis that $\boldsymbol{\beta} = \mathbf{0}$ are as follows:

```
           Variable                                      Standard
     n       Name       Label     Variance     DF         Error

     1       x11        x1 1       0.16667      1       0.40825
     2       x12        x1 2       0.16667      1       0.40825
     3       x21        x2 1       0.16667      1       0.40825
     4       x22        x2 2       0.16667      1       0.40825
     5       x31        x3 1       0.16667      1       0.40825
     6       x32        x3 2       0.16667      1       0.40825
     7       x41        x4 1       0.16667      1       0.40825
     8       x42        x4 2       0.16667      1       0.40825
     9       x51        x5 1       0.16667      1       0.40825
    10       x52        x5 2       0.16667      1       0.40825
    11       x61        x6 1       0.16667      1       0.40825
    12       x62        x6 2       0.16667      1       0.40825
                                               ==
                                               12
```

Since this optimal design and the standardized orthogonal contrast coding is used, the variances are all exactly the same as the *D*-Error, and *D*-Error equals one over the number of choice sets. For nonoptimal designs and with this coding, you would expect that one or more of the variances to exceed one over the number of choice sets. The variances are the diagonal of the covariance matrix. You can examine the covariance matrix for the parameters for the choice model under the assumption that $\boldsymbol{\beta} = 0$ as follows:

```
proc format;
   value zer -1e-12 - 1e-12 = ' 0   ';
   run;

proc print data=bestcov label;
   id __label;
   label __label = '00'x;
   var x:;
   format _numeric_ zer5.2;
   run;
```

The format displays values very close to zero as precisely zero to make a better display.

The results are as follows:

```
          x1 1  x1 2  x2 1  x2 2  x3 1  x3 2  x4 1  x4 2  x5 1  x5 2  x6 1  x6 2

     x1 1  0.17  0     0     0     0     0     0     0     0     0     0     0
     x1 2  0     0.17  0     0     0     0     0     0     0     0     0     0
     x2 1  0     0     0.17  0     0     0     0     0     0     0     0     0
     x2 2  0     0     0     0.17  0     0     0     0     0     0     0     0
     x3 1  0     0     0     0     0.17  0     0     0     0     0     0     0
     x3 2  0     0     0     0     0     0.17  0     0     0     0     0     0
     x4 1  0     0     0     0     0     0     0.17  0     0     0     0     0
     x4 2  0     0     0     0     0     0     0     0.17  0     0     0     0
     x5 1  0     0     0     0     0     0     0     0     0.17  0     0     0
     x5 2  0     0     0     0     0     0     0     0     0     0.17  0     0
     x6 1  0     0     0     0     0     0     0     0     0     0     0.17  0
     x6 2  0     0     0     0     0     0     0     0     0     0     0     0.17
```

The covariance matrix equals $p^{-1}\mathbf{I} = \frac{1}{6}\mathbf{I}$. With an optimal generic design such as this, the covariance matrix is diagonal, and each diagonal value is one over the number of choice sets. Hence, the minimum *D*-error is one over the number of choice sets, and the maximum *D*-efficiency is the number of choice sets. The number of choice sets is used to scale *D*-efficiency to get the relative *D*-efficiency.

The preceding discussion has concerned symmetric designs. A design is said to be *symmetric* when all of the attributes have the same number of levels. When at least one attribute has a different number of levels from at least one other attribute, the design is asymmetric. Designs that are optimal, generic, and asymmetric can also be constructed from orthogonal arrays. For example, using the orthogonal array $2^{27}3^{11}6^{1}12^{1}$ in 72 runs, you can construct an optimal generic choice design with 12 choice sets, 6 alternatives, for attributes $2^{27}3^{11}6^{1}$ as follows:

```
%mktex(12 2 ** 27 3 ** 11 6,        /* Set and factor levels                */
       n=72)                        /* num of sets times num of alts (12x6) */

%mktlab(data=design,                /* design from MktEx                    */
        vars=Set x1-x39)            /* new variable names                   */

%choiceff(data=final,               /* candidate set of choice sets         */
          init=final(keep=set),     /* select these sets from candidates    */
          model=class(x1-x39 / sta),/* model with stdzd orthogonal coding   */
          nsets=12,                 /* 12 choice sets                       */
          nalts=6,                  /* 6 alternatives per set               */
          options=relative,         /* display relative D-efficiency        */
          beta=zero)                /* assumed beta vector, Ho: b=0         */

proc print; by set; id set; var x:; run;
```

The last part of the results and the first part of the design are as follows:

```
                              Final Results

                        Design                    1
                        Choice Sets              12
                        Alternatives              6
                        Parameters               54
                        Maximum Parameters       60
                        D-Efficiency        12.0000
                        Relative D-Eff     100.0000
                        D-Error              0.0833
                        1 / Choice Sets      0.0833

            Variable                                         Standard
       n      Name       Label     Variance     DF             Error

       1      x11        x1 1      0.083333      1           0.28868
       2      x21        x2 1      0.083333      1           0.28868
       3      x31        x3 1      0.083333      1           0.28868
       4      x41        x4 1      0.083333      1           0.28868
       5      x51        x5 1      0.083333      1           0.28868
       6      x61        x6 1      0.083333      1           0.28868
       7      x71        x7 1      0.083333      1           0.28868
       8      x81        x8 1      0.083333      1           0.28868
       9      x91        x9 1      0.083333      1           0.28868
      10      x101       x10 1     0.083333      1           0.28868
      11      x111       x11 1     0.083333      1           0.28868
      12      x121       x12 1     0.083333      1           0.28868
      13      x131       x13 1     0.083333      1           0.28868
      14      x141       x14 1     0.083333      1           0.28868
      15      x151       x15 1     0.083333      1           0.28868
      16      x161       x16 1     0.083333      1           0.28868
      17      x171       x17 1     0.083333      1           0.28868
      18      x181       x18 1     0.083333      1           0.28868
      19      x191       x19 1     0.083333      1           0.28868
      20      x201       x20 1     0.083333      1           0.28868
      21      x211       x21 1     0.083333      1           0.28868
      22      x221       x22 1     0.083333      1           0.28868
      23      x231       x23 1     0.083333      1           0.28868
      24      x241       x24 1     0.083333      1           0.28868
      25      x251       x25 1     0.083333      1           0.28868
      26      x261       x26 1     0.083333      1           0.28868
      27      x271       x27 1     0.083333      1           0.28868
      28      x281       x28 1     0.083333      1           0.28868
      29      x282       x28 2     0.083333      1           0.28868
      30      x291       x29 1     0.083333      1           0.28868
      31      x292       x29 2     0.083333      1           0.28868
      32       x301       x30 1    0.083333      1      0.28868
      33       x302       x30 2    0.083333      1      0.28868
      34       x311       x31 1    0.083333      1      0.28868
      35       x312       x31 2    0.083333      1      0.28868
      36       x321       x32 1    0.083333      1      0.28868
      37       x322       x32 2    0.083333      1      0.28868
      38       x331       x33 1    0.083333      1      0.28868
      39       x332       x33 2    0.083333      1      0.28868
      40       x341       x34 1    0.083333      1      0.28868
      41       x342       x34 2    0.083333      1      0.28868
      42       x351       x35 1    0.083333      1      0.28868
      43       x352       x35 2    0.083333      1      0.28868
      44       x361       x36 1    0.083333      1      0.28868
      45       x362       x36 2    0.083333      1      0.28868
      46       x371       x37 1    0.083333      1      0.28868
      47       x372       x37 2    0.083333      1      0.28868
      48       x381       x38 1    0.083333      1      0.28868
      49       x382       x38 2    0.083333      1      0.28868
      50       x391       x39 1    0.083333      1      0.28868
      51       x392       x39 2    0.083333      1      0.28868
      52       x393       x39 3    0.083333      1      0.28868
      53       x394       x39 4    0.083333      1      0.28868
      54       x395       x39 5    0.083333      1      0.28868
                                                ==
                                                54

S                   x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x
e x x x x x x x x x 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3
t 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  1 2 1 2 2 2 1 1 1 1 2 1 1 2 1 2 2 2 1 1 2 1 2 2 2 1 1 3 3 3 3 3 3 3 3 3 3 3 6
  1 2 2 2 1 1 1 2 1 1 1 2 1 1 2 1 2 2 1 2 1 1 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 5
  2 1 1 1 2 2 2 1 2 2 2 1 2 2 1 2 1 1 2 1 2 2 1 2 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
  2 1 2 1 1 1 2 2 2 2 1 2 2 1 2 1 1 1 2 2 1 2 1 1 1 2 2 3 3 3 3 3 3 3 3 3 3 3 3
  2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 4

2 1 2 1 1 1 2 2 2 1 2 1 1 1 2 2 2 1 2 2 1 2 1 1 1 2 2 2 1 1 1 2 2 3 3 2 2 3 3 1
  1 2 2 1 2 1 1 1 2 2 1 1 1 2 2 2 1 2 2 2 1 2 2 1 2 1 1 2 2 2 3 3 1 1 3 3 1 1 2
  1 2 2 2 1 1 1 2 1 2 2 2 1 1 1 2 1 1 2 1 1 2 1 2 2 2 1 3 3 3 1 1 2 2 1 1 2 2 6
  2 1 1 1 2 2 2 1 2 1 1 1 2 2 2 1 2 2 1 2 2 1 2 1 1 1 2 3 3 3 1 1 2 2 1 1 2 2 3
  2 1 1 2 1 2 2 2 1 1 2 2 2 1 1 1 2 1 1 1 2 1 1 2 1 2 2 2 2 2 3 3 1 1 3 3 1 1 5
  2 1 2 2 2 1 1 1 2 1 2 2 2 1 1 1 2 1 1 2 1 2 2 2 1 1 1 1 1 1 2 2 3 3 2 2 3 3 4
```

```
 3 1 1 2 1 2 2 2 1 1 1 2 1 2 2 2 1 1 1 2 1 1 2 1 2 2 2 1 2 3 3 2 2 3 3 1 1 1 1 5
   1 1 2 2 2 1 2 2 1 2 1 2 1 1 1 2 2 2 2 1 2 2 1 2 1 1 1 3 1 1 3 3 1 1 2 2 2 2 3
   1 2 2 2 1 1 1 2 1 1 2 1 2 2 2 1 1 1 2 1 2 2 2 1 1 1 2 1 2 2 1 1 2 2 3 3 3 3 4
   2 1 1 1 2 2 2 1 2 2 1 2 1 1 1 2 2 2 1 2 1 1 1 2 2 2 1 1 2 2 1 1 2 2 3 3 3 3 1
   2 2 1 1 1 2 1 1 2 1 2 1 2 2 2 1 1 1 1 2 1 1 2 1 2 2 2 3 1 1 3 3 1 1 2 2 2 2 6
   2 2 1 2 1 1 1 2 2 2 1 2 1 1 1 2 2 2 1 2 2 1 2 1 1 1 2 2 3 3 2 2 3 3 1 1 1 1 2
```

The covariance matrix is not displayed because of its size. However, we can get a summary of its values as follows:

```
proc iml;
   use bestcov(keep=x:); read all into x;
   x = round(shape(x,1)‘, 1e-12);
   create veccov from x; append from x;
   quit;

proc freq; run;
```

PROC IML is used to turn the matrix into a vector and round the values. Then PROC FREQ is used to summarize the results. The results are as follows:

```
                          The FREQ Procedure

                                                Cumulative    Cumulative
              COL1    Frequency     Percent     Frequency      Percent
      -----------------------------------------------------------------
                 0        2862       98.15          2862        98.15
      0.0833333333          54        1.85          2916       100.00
```

There are 54 values (the number of parameters) equal to $1/12 \approx 0.0833333333$ (the 54 diagonal values), and the rest (the off-diagonal values) are all zero.

Block Designs

This section discusses balanced incomplete block designs (BIBDs), unbalanced block designs, and incomplete block designs. These are useful in MaxDiff experiments (see page 225) and for making partial profile designs (see page 207). We will begin with a more familiar factorial design example and then show how it is related to a block design. We can use the `%MktEx` macro to make an efficient factorial design with a six-level and a four-level factor in 12 runs as follows:

```
%mktex(6 4,                       /* factor levels                       */
       n=12,                      /* 12 runs                             */
       seed=513,                  /* random number seed                  */
       options=nohistory)         /* do not print iteration history      */
```

The results are as follows:

```
                              The OPTEX Procedure

                            Class Level Information

                           Class  Levels  Values

                           x1       6     1 2 3 4 5 6
                           x2       4     1 2 3 4

                                                                  Average
                                                                 Prediction
    Design                                                        Standard
    Number     D-Efficiency     A-Efficiency     G-Efficiency       Error
    ------------------------------------------------------------------------
         1          87.3580          75.0000         100.0000        0.8660
```

The following step displays the design:

```
proc print; run;
```

The results are as follows:

```
                                 Obs    x1    x2

                                   1     1     2
                                   2     1     3
                                   3     2     1
                                   4     2     2
                                   5     3     3
                                   6     3     4
                                   7     4     2
                                   8     4     4
                                   9     5     1
                                  10     5     3
                                  11     6     1
                                  12     6     4
```

We can transpose this design and display the results as follows:

```
proc transpose data=design out=bd(drop=x1 _:) prefix=b; by x1; run;

proc print; run;
```

The results are as follows:

```
                    Obs    b1    b2

                     1      2     3
                     2      1     2
                     3      3     4
                     4      2     4
                     5      1     3
                     6      1     4
```

In this representation of the design, only the values that were in x2 are displayed, and the values that were in x1 are implicit. They are displayed in the row numbers, the column labeled "Obs" that PROC PRINT displays. This is an example of a block design. The design has $b = 6$ rows or blocks (and the first factor of our factorial design had 6 levels). It has $t = 4$ different values displayed (and the second factor of our factorial design had 4 levels). It has $k = 2$ columns (the $n = kb = 12$ values in x2 divided by $b = 6$ blocks equals $k = 2$ columns). Notice that each of the $t = 4$ treatments occurs exactly $r = 3$ times in the design. Also notice that each of the $4(4 - 1)/2 = 6$ pairs of treatments ((1,2), (1,3), (1,4), (2,3), (2,4), (3,4)) occurs exactly $\lambda = 1$ time. These last two properties mean that this block design is a balanced incomplete block design. It is incomplete in the sense that each block has only a subset of the treatments. In contrast, the following design is complete since each treatment appears in every block:

$$\begin{array}{cccc} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \\ 3 & 4 & 1 & 2 \\ 4 & 3 & 2 & 1 \end{array}$$

While the %MktEx macro can easily find small BIBDs like the one shown in this example, it does not provide the optimal approach to finding BIBDs. It does not use an optimal algorithm, nor does it provide for optimal formatting, processing, or display of the results. Instead, we will use the %MktBIBD macro as follows:

```
%mktbibd(b=6,                  /* 6 blocks                            */
         t=4,                  /* 4 treatments                        */
         k=2,                  /* 2 treatments in each block          */
         seed=350)             /* random number seed                  */
```

The results are as follows:

```
          Block Design Efficiency Criterion      100.0000
          Number of Treatments, t                       4
          Block Size, k                                 2
          Number of Blocks, b                           6
          Treatment Frequency                           3
          Pairwise Frequency                            1
          Total Sample Size                            12
          Positional Frequencies Optimized?           Yes
```

```
               Treatment by Treatment Frequencies

                            1  2  3  4

                         1  3  1  1  1
                         2     3  1  1
                         3        3  1
                         4           3

                Treatment by Position Frequencies

                               1  2

                            1  1  2
                            2  1  2
                            3  2  1
                            4  2  1

                Balanced Incomplete Block Design

                            x1    x2

                             3     2
                             1     4
                             4     3
                             3     1
                             2     1
                             4     2
```

The first line of the output tells us that the design is 100% efficient. This means a BIBD was found. Previously, `%MktEx` found an equivalent design and reported that it was 87.358% efficient. The discrepancy is due to these two values being computed relative to different criteria. The `%MktEx` macro is reporting that relative to the variance matrix that an orthogonal array would have, this design is 87.358% efficient. An orthogonal array cannot exist for the design $6^1 4^1$ in 12 runs since 6×4 does not divide 12. Hence, relative to a hypothetical orthogonal array, these designs are 87.358% efficient, but relative to a BIBD they are 100% efficient.

The first matrix of results shows that each of the 4 treatments occurs 3 times, and each pair of treatments occurs once. The second matrix shows us that the first two treatments both occur in the first position once and in the second position twice. The opposite pattern occurs for the second two treatments. In some BIBDs, every treatment appears in every position the same number of times. In others, such as this one, that is not possible. The positions are optimized after the BIBD is found by the `%MktBIBD` macro. The `%MktEx` macro has no such facility. The last matrix is the BIBD.

The following step creates a block design with $t = 5$ treatments shown in $b = 5$ blocks of size $k = 2$:

```
%mktbibd(b=5,                  /* 5 blocks                        */
         t=5,                  /* 5 treatments                    */
         k=2,                  /* 2 treatments in each block      */
         seed=420)             /* random number seed              */
```

The results are as follows:

```
                    Block Design Efficiency Criterion       89.4427
                    Number of Treatments, t                       5
                    Block Size, k                                 2
                    Number of Blocks, b                           5
                    Average Treatment Frequency                   2
                    Average Pairwise Frequency                  0.5
                    Total Sample Size                            10
                    Positional Frequencies Optimized?           Yes

                           Treatment by Treatment Frequencies

                                      1  2  3  4  5

                                   1  2  0  1  1  0
                                   2     2  1  0  1
                                   3        2  0  0
                                   4           2  1
                                   5              2

                           Treatment by Position Frequencies

                                          1  2

                                       1  1  1
                                       2  1  1
                                       3  1  1
                                       4  1  1
                                       5  1  1

                                         Design

                                       x1    x2

                                        5     4
                                        1     3
                                        2     5
                                        4     1
                                        3     2
```

This is an unbalanced block design. Each treatment occurs the same number of times (twice), but the pairwise frequencies are not equal, so it is not a BIBD. You can also see this by examining the block design efficiency criterion. It is less than 100, so the design is not a BIBD. The treatment by position frequencies, however, are perfect. For many purposes in marketing research, an unbalanced block design is adequate. However, we might be reluctant, depending on our purposes, to use a design where some treatments are never paired with other treatments. Unbalanced block designs occur for many specifications in which a BIBD is not possible.

Next, we will try the same specification again, but this time requesting one fewer block. The following step creates the block design:

```
%mktbibd(b=4,                      /* 4 blocks                            */
         t=5,                      /* 5 treatments                        */
         k=2,                      /* 2 treatments in each block          */
         seed=420)                 /* random number seed                  */
```

The results are as follows:

```
Block Design Efficiency Criterion       74.7674
Number of Treatments, t                       5
Block Size, k                                 2
Number of Blocks, b                           4
Average Treatment Frequency                 1.6
Average Pairwise Frequency                  0.4
Total Sample Size                             8
Positional Frequencies Optimized?           Yes

       Treatment by Treatment Frequencies

                  1  2  3  4  5

               1  2  1  1  0  0
               2     1  0  0  0
               3        2  0  1
               4           1  1
               5              2

       Treatment by Position Frequencies

                      1  2

                   1  1  1
                   2  1  0
                   3  1  1
                   4  0  1
                   5  1  1

                    Design

                   x1    x2

                    1     3
                    2     1
                    3     5
                    5     4
```

This is an incomplete block design. Each treatment does not occur the same number of times. Usually, we would prefer to have a BIBD or an unbalanced block design.

We will use BIBD's (and unbalanced block designs) in two ways in this book. First, they can provide one of the components for constructing a certain class of optimal partial-profile designs (Chrzan and Elrod 1995). In these partial-profile designs, there are t attributes, shown in b blocks of choice sets, where k attributes vary in each block, while the remaining $t - k$ attributes are held constant. That is the subject of the example starting on page 207. Second, they are used for MaxDiff designs. In a MaxDiff study (Louviere 1991, Finn and Louviere 1992), there are t attributes shown in b sets of size k. That is the subject of the example starting on page 225.

You can find the sizes in which a BIBD might be available for ranges of t, b, and k, using the `%MktBSize` macro as in the following example:

```
%mktbsize(t=5,                    /* 5 treatments                         */
        k=2 to 3,                 /* 2 or 3 treatments per block          */
        b=2 to 20)                /* between 2 and 20 blocks              */
```

The results of this step are as follows:

```
                                                                   n
        t             k        b             r            Lambda         Total
        Number of     Block    Number        Treatment    Pairwise       Sample
        Treatments    Size     of Blocks     Frequency    Frequencies    Size

             5          2         10            4              1           20
             5          3         10            6              3           30
```

You can see the unbalanced block designs as follows:

```
%mktbsize(t=5,                    /* 5 treatments                         */
        k=2 to 3,                 /* 2 or 3 treatments per block          */
        b=2 to 20,                /* between 2 and 20 blocks              */
        options=ubd)              /* also show unbalanced block designs   */
```

The results of this step are as follows:

```
                                                                   n
        t             k        b             r            Lambda         Total
        Number of     Block    Number        Treatment    Pairwise       Sample
        Treatments    Size     of Blocks     Frequency    Frequencies    Size

             5          2         5             2             0.5          10
             5          3         5             3             1.5          15
```

Note that these designs are precisely half the size of the designs that were listed previously. By default, the `%MktBSize` macro will not report designs whose sizes are multiples of other reported designs. You can specify `maxreps=2` to get all of the designs as follows:

```
%mktbsize(t=5,                     /* 5 treatments                         */
          k=2 to 3,                /* 2 or 3 treatments per block          */
          b=2 to 20,               /* between 2 and 20 blocks              */
          options=ubd,             /* also show unbalanced block designs   */
          maxreps=2)               /* allow 1 or 2 replications            */
```

The results of this step are as follows:

```
                                                                n
t            k      b           r           Lambda        Total
Number of    Block  Number      Treatment   Pairwise      Sample   Number of
Treatments   Size   of Blocks   Frequency   Frequencies   Size     Replications

     5         2         5           2            0.5        10           1
     5         2        10           4              1        20           2
     5         3         5           3            1.5        15           1
     5         3        10           6              3        30           2
```

The `%MktBSize` macro reports on sizes that meet necessary but not sufficient criteria for the existence of BIBDs. When $r = b \times k/t$ and $l = r \times (k - 1)/(t - 1)$ are integers, and $k = t$ and $b \geq t$, then a complete block design might be possible. When $r = b \times k/t$ and $l = r \times (k - 1)/(t - 1)$ are integers, and $k < t$ and $b \geq t$, then a balanced incomplete block design might be possible. When r is an integer, then an unbalanced block design is possible. The `%MktBIBD` macro will not always find a BIBD even when one is known to exist. However, it usually works quite well in finding BIBDs for small specifications and at least a highly efficient block design for larger specifications.

Both macros have two sets of options for specifying b (`b=`*b* and `nsets=`*b*), t (`t=`*t* and `nattrs=`*t*), and k (`k=`*k* and `setsize=`*k*). When you specify `t=`*t*, you get the output shown previously. The `b=`*b*, `t=`*t*, and `k=`*k* options use a notation that is common for statistical applications. However, when you specify `nattrs=`*t*) the output of both the `%MktBIBD` and `%MktBSize` macros will use the word "Attribute" rather than "Treatment" and "Set" rather than "Block". The following step uses the alternative option names:

```
%mktbsize(nattrs=5,                /* 5 attributes                         */
          setsize=2 to 3,          /* 2 or 3 attributes per set            */
          nsets=2 to 20)           /* between 2 and 20 sets                */
```

The results of this step are as follows:

```
                                                                    n
   t             k        b            r             Lambda         Total
   Number of     Set      Number       Attribute     Pairwise       Sample
   Attributes    Size     of Sets      Frequency     Frequencies    Size

        5          2        10            4              1            20
        5          3        10            6              3            30
```

The Process of Designing a Choice Experiment

It is important that you understand a number of things in this chapter before you design your first choice experiment. Most of this chapter is fairly straight-forward, but without a clear understanding of it, you will no doubt get confused when you actually design an experiment. You should go back and review this chapter if you are not completely comfortable with the meaning of any of these terms: linear arrangement, choice design, generic choice design, factors, attributes, alternatives, choice sets, orthogonality, balance, and efficiency. In particular, the meaning of *linear arrangement* and *choice design* (pages 67–71) and the relationship between the two is fundamental. These two design layouts are the source of a great deal of confusion when many people start out. Make sure that you understand them. You do not have to understand the formula for the variance matrix for a choice model, the orthogonal coding, or the formulas for efficiency. However, you should be comfortable with the idea of the average variability of the parameter estimates and how it is related to efficiency.

This section lists the steps in designing a choice experiment. The next section illustrates these steps with several simple examples. You should work through the simple examples in the next section before consulting the more complex examples in the discrete choice chapter on page 285.

The first step in designing a choice experiment involves determining:

- Is this a generic study (no brands) or a branded study? Branded studies have a label for each alternative that conveys meaning beyond ordinary attributes. Brand names are the most common example. See pages 127, 166, 188, and 302 for examples of studies with brands. The destinations in the vacation example (pages 339 and 410) also act like brands. In a generic study, the alternatives are simply bundles of attributes. See pages 198, 102, and 556 for examples of generic designs. Also see the documentation for the `%ChoicEff` macro beginning on page 806 for examples of generic design construction.
- If it is branded, what are the brands?
- How many alternatives?
- Is there a constant (none, no purchase, delay purchase, or stick with my regular brand) alternative?
- What are the attributes of all of the alternatives, and what are their levels?
- Are any of the attributes generic? In other words, are there attributes that you expect to behave the same way across all alternatives?

- Are any of the attributes alternative-specific? In other words, are there attributes that you expect to behave differently across all alternatives (brand by attribute interactions)?
- Are there any restrictions, within alternatives, within choice sets, or across choice sets?

Step 1. Write down all of the attributes of all of your alternatives and their levels. See pages 339, 410, 469, and 556 for examples.

Step 2. Is this a generic study (such as the chair example on page 556) or a branded example (such as the vacation examples on pages 339 and 410 and the food example on page 469)?

Step 3. If this is a branded study:

- Use the `%MktRuns` macro to suggest the number of choice sets. See page 1159 for documentation. See the following pages for examples of using this macro in this chapter: 128 and 188. Also see the following pages for examples of using this macro in the discrete choice chapter: 340, 411, 415, 482, and 483. In addition, see the following pages for examples of using this macro in the macro documentation chapter: 895, 905, 916, 919, 1057, 1159, 1160, 1161, 1162, and 1165.
- Use the `%MktEx` macro to make a linear arrangement of a choice design. See page 1017 for documentation. See the following pages for examples of using this macro in this chapter: 129 and 190. Also see the following pages for examples of using this macro in the discrete choice chapter: 304, 304, 320, 333, 343, 352, 413, 415, 416, 417, 422, 425, 472, 479, 485, 488, and 491. In addition, see the following pages for examples of using this macro in the macro documentation chapter: 804, 808, 815, 817, 819, 858, 878, 881, 896, 903, 905, 917, 919, 920, 921, 923, 924, 926, 926, 927, 929, 952, 979, 979, 1002, 1005, 1006, 1008, 1009, 1012, 1017, 1018, 1026, 1027, 1028, 1029, 1030, 1030, 1058, 1062, 1067, 1067, 1068, 1069, 1073, 1076, 1079, 1080, 1082, 1093, 1093, 1093, 1096, 1098, 1099, 1102, 1103, 1135, 1137, 1142, 1142, 1142, 1143, 1145, 1150, 1154, 1154, and 1155.
- Use the `%MktEval` macro to evaluate the linear arrangement. See page 1012 for documentation. See page 130 for an example of using this macro in this chapter. Also see the following pages for examples of using this macro in the discrete choice chapter: 306, 308, 349, 353, 413, 423, 480, 485, 489, 491, 493, 538, 588, and 591. In addition, see the following pages for examples of using this macro in the macro documentation chapter: 959, 1012, 1073, and 1099.
- Display and check the linear arrangement. See page 305 for an example.
- Use the `%MktKey` and `%MktRoll` macros to make a choice design from the linear arrangement. See page 1153 for documentation on the `%MktRoll` macro and page 1090 for documentation on the `%MktKey` macro. See the following pages for examples of using the `%MktRoll` macro in this chapter: 134 and 192. Also see the following pages for examples of using this macro in the discrete choice chapter: 312, 320, 357, 387, 429, 505, 546, 556, 575, 607, 617, 628, and 636. In addition, see the following pages for examples of using this macro in the macro documentation chapter: 819, 878, 881, 898, 917, 929, 982, 1005, 1008, 1085, 1154, 1155, 1156, and 1156. See the following pages for examples of using the `%MktKey` macro in this chapter: 133 and 192. Also see the following pages for examples of using this macro in the discrete choice chapter: 356, 546, 556, 575, 607, 617, 628, and 636. In addition, see the following pages for examples of using this macro in the macro documentation chapter: 897, 1085, 1090, 1090, 1090, and 1153.

Step 4. If this is a generic study:

- Use the `%MktRuns` macro to suggest a size for the candidate design. See page 1159 for documentation. See page 199 for an example of using this macro in this chapter. Also see page 557 for an example of using this macro in the discrete choice chapter. In addition, see the following pages for examples of using this macro in the macro documentation chapter: 895, 905, 916, 919, 1057, 1159, 1160, 1161, 1162, and 1165.

- Use the `%MktEx` macro to make a candidate design. See page 1017 for documentation. See the following pages for examples of using this macro in this chapter: 81, 85, 166, 200, 109, 112, and 98. Also see the following pages for examples of using this macro in the discrete choice chapter: 556, 558, 564, 567, 570, and 575. In addition, see the following pages for examples of using this macro in the macro documentation chapter: 804, 808, 815, 817, 819, 858, 878, 881, 896, 903, 905, 917, 919, 920, 921, 923, 924, 926, 926, 927, 929, 952, 979, 979, 1002, 1005, 1006, 1008, 1009, 1012, 1017, 1018, 1026, 1027, 1028, 1029, 1030, 1030, 1058, 1062, 1067, 1067, 1068, 1069, 1073, 1076, 1079, 1080, 1082, 1093, 1093, 1093, 1096, 1098, 1099, 1102, 1103, 1135, 1137, 1142, 1142, 1142, 1143, 1145, 1150, 1154, 1154, and 1155.

- Use the `%MktLab` macro to add alternative flags. See page 1093 for documentation. See the following pages for examples of using this macro in this chapter: 85 and 201. Also see the following pages for examples of using this macro in the discrete choice chapter: 564, and 567. In addition, see the following pages for examples of using this macro in the macro documentation chapter: 814, 815, 817, 858, 921, 926, 952, 979, 1006, 1062, 1093, 1094, 1095, 1096, 1098, 1099, 1102, and 1103.

- Display and check the candidate design. See page 201 for an example.

- Use the `%ChoicEff` macro to find an efficient choice design. See page 806 for documentation. See the following pages for examples of using this macro in this chapter: 87, 170, 193, and 203. Also see the following pages for examples of using this macro in the discrete choice chapter: 320, 559, 564, 567, 570, 570, 574, 576, 607, 618, 628, 632, and 636. In addition, see the following pages for examples of using this macro in the macro documentation chapter: 809, 816, 817, 819, 859, 862, 863, 878, 880, 882, 883, 887, 891, 899, 901, 908, 913, 917, 919, 920, 921, 924, 926, 926, 927, 929, 952, 982, 1005, 1006, 1145, and 1150.

- Display and check the choice design. See page 205 for an example.

- Go back and try the `%MktEx` step with other size choice sets (unless you are using a small, full-factorial candidate set). Stop when you feel comfortable with the results.

Step 5. Continue processing the design:

- Display and check the choice design. See page 134 for an example.

- Assign formats and labels. See page 136 for an example.

- Display and check the choice design. See page 136 for an example.

- Use the `%ChoicEff` macro to evaluate the design. See page 806 for documentation. See the following pages for examples of using this macro in this chapter: 81, 83, 137, 140, 142, 109, and 112. Also see the following pages for examples of using this macro in the discrete choice chapter: 313, 317, 322, 360, 365, 366, 430, 508, 509, 542, 570, 574, 597, 599, 645, 650, 654, 656, 659, and 662. In addition, see the following pages for examples of using this macro in the macro documentation chapter: 809, 816, 817, 819, 859, 862, 863, 878, 880, 882, 883, 887, 891, 899, 901, 908, 913, 917, 919, 920, 921, 924, 926, 926, 927, 929, 952, 982, 1005, 1006, 1145, and 1150.

- Use the `%MktDups` macro to check for duplicate choice sets. See page 1004 for documentation. See the following pages for examples of using this macro in this chapter: 147, 174, 198, and 206. Also see the following pages for examples of using this macro in the discrete choice chapter: 319, 368, 519, 564, 564, 567, 570, 576, 597, 607, 617, 628, 636, 645, 650, 654, 656, 659, and 662. In addition, see the following pages for examples of using this macro in the macro documentation chapter: 809, 817, 819, 1005, 1006, 1008, 1009, and 1010.

- For larger designs, you might need to block the design. See page 979 for documentation. Also see the following pages for examples of using this macro in the discrete choice chapter: 426 and 497. In addition, see the following pages for examples of using this macro in the macro documentation chapter: 979, 982, 1098, and 1099. Also see the following pages for examples of using this macro in the discrete choice chapter: 641, 642, and 660. Alternatively, with the linear arrangement, you can sometimes just add a blocking factor directly to the linear arrangement. See page 979 for an example.

Step 6. Collect data and process the design:

- Display or otherwise generate the choice tasks, and then collect and enter the data. See page 147 for an example.

- Use the `%MktMerge` macro to merge the data and the design. See page 1125 for documentation. See the following pages for examples of using this macro in this chapter: 149 and 176. Also see the following pages for examples of using this macro in the discrete choice chapter: 325, 371, 387, 437, 522, and 529. In addition, see the following pages for examples of using this macro in the macro documentation chapter: 1125, 1125, and 1126.

- Display part of the data and design and check the results. See page 149 for an example.

- Optionally, particularly for large data sets, you can aggregate the data set using PROC SUMMARY. See page 522 for an example.

- Use the TRANSREG procedure to code the design. See the following pages for examples of using this procedure: 150, 176, 327, 372, 378, 380, 383, 388, 438, 447, 449, 452, 460, 460, 462, 464, 523, 528, 530, and 528.

- Display part of the coded design and check the results. See page 150 for an example.

- Use the `%PHChoice` macro to customize the output. See page 1173 for documentation. See the following pages for examples of using this macro in this chapter: 152 and 187. Also see the following pages for examples of using this macro in the discrete choice chapter: 287 and 288. In addition, see the following pages for examples of using this macro in the macro documentation chapter: 1173, 1173, and 1177.

- Use the PHREG procedure to fit the multinomial logit model. See pages 152, 176, 295, 298, 329, 375, 378, 380, 385, 390, 440, 442, 447, 449, 457, 462, 464, 524, 528, 532, 550, and 552.

There are many variations not covered in this simple outline. See the examples in the discrete choice chapter (pages 285–663) for many other possibilities.

Overview of the Examples

The next six sections show how to create small and simple choice experiments from start to finish. Each example illustrates one of the basic approaches to making a choice design. These examples are simple. In contrast, the examples on pages 285 through 663 tend to be much more involved and have many more nuances. These introductory examples show the basic steps in the context of a design with no complications. Note, however, that each of these examples is longer than it needs to be because each displays extra information to help you better understand how choice designs are created and how they work. Also note that steps that are common to all examples are explained in more detail in the earlier examples than in the later examples, so please read all of them to get a full understanding of the process. Understanding these introductory examples will help you with the more involved examples that come later. The first example directly uses the `%MktEx` macro to find a design. The other examples use a combination of the `%MktEx` macro, the `%ChoicEff` macro, and in some cases, other macros.

Example 1: Orthogonal and Balanced Factors, the Linear Arrangement Approach

In this example, we use the `%MktEx` macro to find a linear arrangement of a choice design, then we convert it into choice design format. You should use this approach when you want all of the attributes of all of the alternatives to be balanced and orthogonal (as in this example) or at least nearly so (as in most real-life examples that are more complicated). This approach lets you fit complicated models including models with alternative-specific effects without specifying in advance the exact nature of the model or parameters.

The product is breakfast bars, and there are three brands, Branolicious, Brantopia, and Brantasia.[*] The choice sets consist of three brands and a constant (no purchase) alternative. Each brand has two attributes, a four-level attribute for price and a two-level attribute for the number of bars per box. The prices are \$2.89, \$2.99, \$3.09, and \$3.19, and the sizes are 6 count and 8 count. We can make a choice design by first making a design that is optimal for a hypothetical linear model that has factors for all of the attributes of all of the alternatives. The linear arrangement consists of the six factors, which are shown organized by brand and also organized by attribute. There is only one set of attributes, however that set is shown in two different ways. The attributes are as follows:

Factors Organized By Brand

Linear Factor Name	Levels	Brand	Choice Design Attribute
x1	4 levels	Branolicious	Price
x2	2 levels	Branolicious	Count
x3	4 levels	Brantopia	Price
x4	2 levels	Brantopia	Count
x5	4 levels	Brantasia	Price
x6	2 levels	Brantasia	Count

Factors Organized By Attribute

Linear Factor Name	Levels	Brand	Choice Design Attribute
x1	4 levels	Branolicious	Price
x3	4 levels	Brantopia	Price
x5	4 levels	Brantasia	Price
x2	2 levels	Branolicious	Count
x4	2 levels	Brantopia	Count
x6	2 levels	Brantasia	Count

[*]Real studies, of course, use real brands. Since we have not collected real data, we cannot use real brand names. We picked these silly names so no one would confuse our artificial data with real data.

We need a factorial design with 6 factors: Branolicious Price, Branolicious Count, Brantopia Price, Brantopia Count, Brantasia Price, and Brantasia Count. From it, we make a choice design with three attributes, brand, count, and price. We can use the `%MktRuns` macro as follows to suggest the number of choice sets:

```
title 'Cereal Bars';

%mktruns(4 2  4 2  4 2)     /* factor level list for all attrs and alts     */
```

The input to the macro is the number of levels of all of the factors (that is, all of the attributes of all of the alternatives). The output from the macro is as follows:

```
                                 Cereal Bars

                               Design Summary

                           Number of
                           Levels       Frequency

                                2           3
                                4           3

                                 Cereal Bars

              Saturated      = 13
              Full Factorial = 512

              Some Reasonable                      Cannot Be
                 Design Sizes       Violations     Divided By

                           16 *              0
                           32 *              0
                           24                3     16
                           20               12      8 16
                           28               12      8 16
                           14               18      4  8 16
                           18               18      4  8 16
                           22               18      4  8 16
                           26               18      4  8 16
                           30               18      4  8 16
                           13 S             21      2  4  8 16

            * - 100% Efficient design can be made with the MktEx macro.
            S - Saturated Design - The smallest design that can be made.
                Note that the saturated design is not one of the
                recommended designs for this problem.  It is shown
                to provide some context for the recommended sizes.
```

```
                                 Cereal Bars

       n                    Design                       Reference

      16    2 **  6            4 **  3                Fractional-Factorial
      16    2 **  3            4 **  4                Fractional-Factorial
      32    2 ** 22            4 **  3                Fractional-Factorial
      32    2 ** 19            4 **  4                Fractional-Factorial
      32    2 ** 16            4 **  5                Fractional-Factorial
      32    2 ** 15            4 **  3   8 **  1      Fractional-Factorial
      32    2 ** 13            4 **  6                Fractional-Factorial
      32    2 ** 12            4 **  4   8 **  1      Fractional-Factorial
      32    2 ** 10            4 **  7                Fractional-Factorial
      32    2 **  9            4 **  5   8 **  1      Fractional-Factorial
      32    2 **  7            4 **  8                Fractional-Factorial
      32    2 **  6            4 **  6   8 **  1      Fractional-Factorial
      32    2 **  4            4 **  9                Fractional-Factorial
      32    2 **  3            4 **  7   8 **  1      Fractional-Factorial
```

The output tells us that there are 3 two-level factors and 3 four-level factors. The saturated design has 13 runs or rows, so we need at least 13 choice sets with this approach. The full-factorial design has 512 runs, so there are a maximum of 512 possible choice sets. The `%MktRuns` macro suggests 16 as its first choice because 16 meets necessary but not sufficient conditions for the existence of an orthogonal array. Sixteen can be divided by 2 (we have two-level factors), 4 (we have four-level factors), 2×2 (we have more than one two-level factor), 4×4 (we have more than one four-level factor), and 2×4 (we have both two-level factors and four-level factors). The number of choice sets must be divisible by all of these if the design is going to be orthogonal and balanced. Thirty-two meets these conditions as well. However, 16 is a more reasonable number of judgments for people to make, and the other suggestions (24, 20, 28, 14, 18, 22, 26, 30) all cannot be divided by at least one of the relevant numbers. For this example, the macro only considers sizes up to 32. By default, the macro stops considering larger sizes when it finds a perfect size (in this case 32) that is twice as big as another perfect size (16). Sixteen choice sets is ideal for this example. The necessary conditions are sufficient in this case, and there is an orthogonal-array that we can use. The last part of the output lists the orthogonal arrays that `%MktEx` knows how to make that work for our specification.

We use the `%MktEx` macro as follows to get our factorial design as follows:

```
%mktex(4 2  4 2  4 2,       /* factor level list for all attrs and alts     */
       n=16,                /* number of choice sets                        */
       seed=17)             /* random number seed                           */
```

The macro accepts a factor-level list like the `%MktRuns` list along with the number of runs or choice sets. We specify a random number seed so that we always get the same design if we rerun the `%MktEx` macro.

The results are as follows:

```
                                 Cereal Bars

                           Algorithm Search History

                                Current          Best
              Design    Row,Col  D-Efficiency  D-Efficiency  Notes
              ----------------------------------------------------------
                   1      Start      100.0000      100.0000  Tab
                   1        End      100.0000
                                 Cereal Bars

                             The OPTEX Procedure

                           Class Level Information

                            Class  Levels  Values
                            x1        4     1 2 3 4
                            x2        2     1 2
                            x3        4     1 2 3 4
                            x4        2     1 2
                            x5        4     1 2 3 4
                            x6        2     1 2


                                 Cereal Bars
                                                                  Average
                                                                Prediction
       Design                                                    Standard
       Number     D-Efficiency     A-Efficiency     G-Efficiency       Error
       ------------------------------------------------------------------------
            1       100.0000         100.0000         100.0000          0.9014
```

The `%MktEx` macro found a 100% efficient, orthogonal and balanced design with 3 two-level factors and 3 four-level factors, just as the `%MktRuns` macro told us it would. The levels are all positive integers, starting with 1 and continuing up to the number of levels. The note in the algorithm search history of "`Tab`" on a line that displays 100% efficiency shows that the design was directly constructed from the `%MktEx` macro's table or catalog of orthogonal designs.

Next, we examine some of the properties of the design and display it. This step is not necessary since we have a 100% efficient design. However, we go through it here to better see the properties of a 100% efficient design. The `%MktEval` macro tells us which factors are orthogonal and which are correlated. It also tells us how often each level occurs, how often each pair of levels occurs across pairs of factors, and how often each run or choice set occurs. The following steps evaluate and display the design:

```
title2 'Examine Correlations and Frequencies';

%mkteval(data=randomized)   /* evaluate randomized design                 */

title2 'Examine Design';
proc print data=randomized; run;
```

The first part of the output is as follows:

```
                              Cereal Bars
                  Examine Correlations and Frequencies
                Canonical Correlations Between the Factors
           There are 0 Canonical Correlations Greater Than 0.316

                   x1      x2      x3      x4      x5      x6

            x1     1       0       0       0       0       0
            x2     0       1       0       0       0       0
            x3     0       0       1       0       0       0
            x4     0       0       0       1       0       0
            x5     0       0       0       0       1       0
            x6     0       0       0       0       0       1
```

All canonical correlations off the diagonal are zero, which tells us that the design is orthogonal—that every factor is uncorrelated with every other factor.

The next part of the output is as follows:

```
                              Cereal Bars
                  Examine Correlations and Frequencies
                         Summary of Frequencies
           There are 0 Canonical Correlations Greater Than 0.316

                            Frequencies

                   x1       4 4 4 4
                   x2       8 8
                   x3       4 4 4 4
                   x4       8 8
                   x5       4 4 4 4
                   x6       8 8
                   x1 x2    2 2 2 2 2 2 2 2
                   x1 x3    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                   x1 x4    2 2 2 2 2 2 2 2
                   x1 x5    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                   x1 x6    2 2 2 2 2 2 2 2
```

```
x2 x3    2 2 2 2 2 2 2 2
x2 x4    4 4 4 4
x2 x5    2 2 2 2 2 2 2 2
x2 x6    4 4 4 4
x3 x4    2 2 2 2 2 2 2 2
x3 x5    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
x3 x6    2 2 2 2 2 2 2 2
x4 x5    2 2 2 2 2 2 2 2
x4 x6    4 4 4 4
x5 x6    2 2 2 2 2 2 2 2
N-Way    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

This is a very compact presentation of output from PROC SUMMARY. The one-way, two-way, and *n*-way frequencies show us how often each level, pair of levels, and choice set occurs. It tells us that each level occurs equally often, (4 times in the four-level factors and 8 times in the two-level factors), and each pair of levels occurs equally often. The *n*-way frequencies tell us that every choice set occurs only once in the design—there are no duplicate choice sets.

The randomized design is follows:

```
                    Cereal Bars
                  Examine Design

Obs    x1    x2    x3    x4    x5    x6

  1     1     1     2     1     2     1
  2     4     1     2     2     4     2
  3     3     2     2     1     3     2
  4     3     1     4     1     4     1
  5     2     2     2     2     1     1
  6     4     1     1     1     1     1
  7     3     2     1     2     2     1
  8     1     2     3     2     4     1
  9     2     1     4     2     2     2
 10     1     2     4     1     1     2
 11     4     2     4     2     3     1
 12     2     1     3     1     3     1
 13     2     2     1     1     4     2
 14     1     1     1     2     3     2
 15     3     1     3     2     1     2
 16     4     2     3     1     2     2
```

It has 3 four-level factors with levels 1, 2, 3, 4, and 3 two-level factors with levels 1 and 2. It has 16 rows since there are 16 choice sets. The levels and rows are not sorted (that is, the design is randomized), so this linear arrangement is in a good form to use to make the choice design.

Next, we need to make a choice design from our linear arrangement. This involves taking levels for alternatives, which are next to each other in the linear arrangement, and moving them on top of each other to form the choice design. We specify the rules for doing this in a SAS data set. We call this data set the key to constructing the choice design. Before we describe how this data set is created and what it means, let's look at the key data set to see where we are going. The following output displays the key data set for this example:

```
Obs    Brand           Price    Count

 1     Branolicious     x1       x2
 2     Brantopia        x3       x4
 3     Brantasia        x5       x6
 4     None
```

We need to specify that the brands are Branolicious, Brantopia, Brantasia, and None. We need to specify that the Branolicious Price is made from x1, the Branolicious Count is made from x2, the Brantopia Price is made from x3, the Brantopia Count is made from x4, the Brantasia Price is made from x5, and the Brantasia Count is made from x6. We also need to specify that the None alternative is not made from any of the attributes. The variables in this data set correspond to the attributes in the choice design, and the values correspond to the brands and to the factorial design factors. The `%MktKey` macro gives us the linear arrangement factor names that we can copy and paste into this data set. For many designs (particularly larger designs), this macro makes it easy to construct the design key. The following step creates the names:

```
%mktkey(3 2)              /* x1-x6 (since 3*2=6) in 3 rows and 2 columns  */
```

The results are as follows:

```
x1    x2

x1    x2
x3    x4
x5    x6
```

The names `x1-x6` are arranged into three rows (the first value of "3 2") and two columns (the second value of "3 2") for pasting into the key data set. The following step creates the key data set:

```
title2 'Create the Choice Design Key';

data key;
   input
Brand $ 1-12    Price $    Count $; datalines;
Branolicious    x1         x2
Brantopia       x3         x4
Brantasia       x5         x6
None            .          .
;
```

Note that when reading missing or blank character data with list input in a DATA step, as we do here, you can use a period for the blank values. SAS automatically translates them into blanks.

The `%MktRoll` macro processes the linear arrangement using the information in the key data set to make the choice design as follows:

```
title2 'Create Choice Design from Linear Arrangement';

%mktroll(design=randomized,  /* input randomized linear arrangement         */
         key=key,            /* rules for making choice design              */
         alt=brand,          /* brand or alternative label var              */
         out=cerealdes)      /* output choice design                        */

proc print; id set; by set; run;
```

The choice design contains the variable `Set` along with the variable names and brands from the key data set. The information from the linear arrangement is all stored in the right places. The `Brand` variable contains literal names, and it is named in the `alt=` option, which designates the alternative name (often brand) attribute. The remaining variables contain factor names from the linear data set. The choice design is as follows:

```
                              Cereal Bars
                 Create Choice Design from Linear Arrangement

                    Set    Brand           Price    Count

                      1    Branolicious      1        1
                           Brantopia         2        1
                           Brantasia         2        1
                           None              .        .

                      2    Branolicious      4        1
                           Brantopia         2        2
                           Brantasia         4        2
                           None              .        .

                      3    Branolicious      3        2
                           Brantopia         2        1
                           Brantasia         3        2
                           None              .        .

                      4    Branolicious      3        1
                           Brantopia         4        1
                           Brantasia         4        1
                           None              .        .

                      5    Branolicious      2        2
                           Brantopia         2        2
                           Brantasia         1        1
                           None              .        .
```

```
  6    Branolicious      4       1
       Brantopia         1       1
       Brantasia         1       1
       None              .       .

  7    Branolicious      3       2
       Brantopia         1       2
       Brantasia         2       1
       None              .       .

  8    Branolicious      1       2
       Brantopia         3       2
       Brantasia         4       1
       None              .       .

  9    Branolicious      2       1
       Brantopia         4       2
       Brantasia         2       2
       None              .       .

 10    Branolicious      1       2
       Brantopia         4       1
       Brantasia         1       2
       None              .       .

 11    Branolicious      4       2
       Brantopia         4       2
       Brantasia         3       1
       None              .       .

 12    Branolicious      2       1
       Brantopia         3       1
       Brantasia         3       1
       None              .       .

 13    Branolicious      2       2
       Brantopia         1       1
       Brantasia         4       2
       None              .       .

 14    Branolicious      1       1
       Brantopia         1       2
       Brantasia         3       2
       None              .       .

 15    Branolicious      3       1
       Brantopia         3       2
       Brantasia         1       2
       None              .       .
```

```
           16    Branolicious      4        2
                 Brantopia         3        1
                 Brantasia         2        2
                 None              .        .
```

These following steps assign formats to the levels and display the choice sets:

```
title2 'Final Choice Design';

proc format;
   value price 1 = $2.89 2 = $2.99 3 = $3.09 4 = $3.19 . = ' ';
   value count 1 = 'Six Bars' 2 = 'Eight Bars'  . = ' ';
   run;

data sasuser.cerealdes;
   set cerealdes;
   format price price. count count.;
   run;

proc print data=sasuser.cerealdes(obs=16);
   by set;   id set;
   run;
```

In the interest of space, only the first four choice sets are displayed. The design is stored in a permanent SAS data set so it is available at analysis time. The first four choice sets are as follows:

```
                        Cereal Bars
                    Final Choice Design

           Set    Brand            Price      Count

            1     Branolicious     $2.89    Six Bars
                  Brantopia        $2.99    Six Bars
                  Brantasia        $2.99    Six Bars
                  None

            2     Branolicious     $3.19    Six Bars
                  Brantopia        $2.99    Eight Bars
                  Brantasia        $3.19    Eight Bars
                  None

            3     Branolicious     $3.09    Eight Bars
                  Brantopia        $2.99    Six Bars
                  Brantasia        $3.09    Eight Bars
                  None
```

```
         4     Branolicious    $3.09    Six Bars
               Brantopia       $3.19    Six Bars
               Brantasia       $3.19    Six Bars
               None
```

The following step evaluates the goodness of the design for a choice model using the `%ChoicEff` macro:

```
title2 'Evaluate Design';

%choiceff(data=sasuser.cerealdes,          /* candidate choice sets         */
          init=sasuser.cerealdes(keep=set),/* select these sets from cands  */
          intiter=0,                       /* eval without internal iters   */
          model=class(brand price count),  /* model, ref cell coding        */
          nalts=4,                         /* number of alternatives        */
          nsets=16,                        /* number of choice sets         */
          beta=zero)                       /* assumed beta vector, Ho: b=0  */
```

The `%ChoicEff` macro constructs the covariance matrix of the specified choice model parameters and displays the variances and standard errors. It also displays the *D*-efficiency and other information. Here, we are using the `%ChoicEff` macro to evaluate a design. It can also be used to search for efficient choice designs. When we evaluate a design, we need to provide the design in the `data=` specification. Usually, you use the `data=` option to specify the candidate set to be searched. In some sense, the `data=` design is a candidate set in this context as well, and we use the `init=` option to specify how the final design is constructed from the candidate set. We do this by bringing in just the choice set numbers in the initial design. This is accomplished with the `init=sasuser.cerealdes(keep=set)` specification. Then the `%ChoicEff` macro selects just the specified candidate choice sets (in this case all of them) and uses them as the initial design. The `intiter=0` option specifies no internal iterations, so the design is evaluated but no attempt is made to improve upon it. Other options include a specification of the number of alternatives, the number of choice sets, and the assumed beta vector. You can specify a list of parameter values or `beta=zero` for all zeros.

The first part of the output is as follows:

```
                                Cereal Bars
                              Evaluate Design

                n     Name                  Beta    Label

                1     BrandBranolicious      0      Brand Branolicious
                2     BrandBrantasia         0      Brand Brantasia
                3     BrandBrantopia         0      Brand Brantopia
                4     Price_2_89             0      Price $2.89
                5     Price_2_99             0      Price $2.99
                6     Price_3_09             0      Price $3.09
                7     CountSix_Bars          0      Count Six Bars
```

This table provides a list of the generated names for all of the parameters, the specified beta value for each, and the generated label. Whenever you specify a list of betas, you need to use this table to ensure that the right betas are assigned to the right parameters.

The next part of the output is as follows:

```
                              Cereal Bars
                            Evaluate Design

              Design   Iteration  D-Efficiency       D-Error
              ----------------------------------------------
                  1        0           1.93756       0.51611
```

This part of the output contains the iteration history table. Since `intiter=0` was specified, this contains only a report of the efficiency of the initial design, which is labeled as iteration 0.

The following results contain the last output tables, which are what we are most interested in seeing:

```
                              Cereal Bars
                            Evaluate Design

                             Final Results

                        Design                  1
                        Choice Sets            16
                        Alternatives            4
                        Parameters              7
                        Maximum Parameters     48
                        D-Efficiency       1.9376
                        D-Error            0.5161

                              Cereal Bars
                            Evaluate Design

                                                                   Standard
      n    Variable Name        Label                 Variance    DF     Error

      1    BrandBranolicious    Brand Branolicious     0.94444     1     0.97183
      2    BrandBrantasia       Brand Brantasia        0.94444     1     0.97183
      3    BrandBrantopia       Brand Brantopia        0.94444     1     0.97183
      4    Price_2_89           Price $2.89            0.88889     1     0.94281
      5    Price_2_99           Price $2.99            0.88889     1     0.94281
      6    Price_3_09           Price $3.09            0.88889     1     0.94281
      7    CountSix_Bars        Count Six Bars         0.44444     1     0.66667
                                                                  ==
                                                                   7
```

We see three parameters for brand (4 alternatives including None minus 1), three for price $(4-1)$, one for count $(2-1)$. All are estimable, and all have reasonable standard errors. With 16 choice sets and 4 alternatives, we can estimate at most $16 \times (4-1) = 48$ parameters. Note that with the `%ChoicEff` macro and with many choice designs and coding schemes, *D*-efficiency is not reported on a 0 to 100 scale as it is in the `%MktEx` macro and linear model designs. This is because the range over which *D*-efficiency can vary is less clear with some choice designs. However, later in this example, we will come close. Also see page 102 for examples of choice designs with *D*-efficiency scaled to the 0 to 100 range. For now, we know the following:

- *D*-efficiency is 1.9376 on a scale of 0 to unknown.
- All parameters are estimable and the design is more than big enough to estimate all of our parameters.
- The variances (and standard errors) are constant within each attribute, which is usually a good sign.
- The variances (and standard errors) are all of a similar magnitude, which is usually a good sign. When the variances vary a lot (which is hard to quantify, but varying by a factor of over 100 or 1000 is certainly enough to make you worry) it is usually a sign of a problem with the design. Often it is a sign of too few choice sets to precisely estimate all of the parameters.

This pattern of variances and standard errors suggests (but certainly does not prove) that this is a good design. We can run the macro again and use the standardized orthogonal contrast coding to get a better evaluation this design. However, before we do that, it is instructive to look at the covariance matrix of the parameter estimates. The following steps display this matrix, which is automatically output to a SAS data set called `bestcov`:

```
proc format;
   value zer -1e-12 - 1e-12 = ' 0   ';
   run;

proc print data=bestcov label;
   id __label;
   label __label = '00'x;
   var BrandBranolicious -- CountSix_Bars;
   format _numeric_ zer5.2;
   run;
```

The format simply displays values very close to zero, both above and below zero, as precisely zero to make a better display.

The results are as follows:

```
                                     Cereal Bars
                                   Evaluate Design

                          Brand       Brand     Brand    Price Price Price  Count
                        Branolicious Brantasia Brantopia $2.89 $2.99 $3.09 Six Bars

    Brand Branolicious     0.94        0.69      0.69    -0.44 -0.44 -0.44  -0.22
    Brand Brantasia        0.69        0.94      0.69    -0.44 -0.44 -0.44  -0.22
    Brand Brantopia        0.69        0.69      0.94    -0.44 -0.44 -0.44  -0.22
    Price $2.89           -0.44       -0.44     -0.44     0.89  0.44  0.44   0
    Price $2.99           -0.44       -0.44     -0.44     0.44  0.89  0.44   0
    Price $3.09           -0.44       -0.44     -0.44     0.44  0.44  0.89   0
    Count Six Bars        -0.22       -0.22     -0.22    0     0     0       0.44
```

You can see that the diagonal of the covariance matrix contains the variances that are reported by the `%ChoicEff` macro. The off-diagonal elements show the covariances. The variances and covariances depend on how the design is coded. Here, the default reference-cell coding is used. Other coding schemes will create quite different results. The standardized orthogonal contrast coding is of particular interest when you evaluate designs. See page 73. The `sta`[*] (short for `standorth`) option in the `model=` option requests a standardized orthogonal contrast coding. The following steps generate and display the covariance matrix with the standardized orthogonal contrast coding:

```
%choiceff(data=sasuser.cerealdes,          /* candidate choice sets         */
          init=sasuser.cerealdes(keep=set),/* select these sets from cands  */
          intiter=0,                       /* eval without internal iters   */
                                           /* model with stdz orthog coding */
          model=class(brand price count / sta),
          nalts=4,                         /* number of alternatives        */
          nsets=16,                        /* number of choice sets         */
          options=relative,                /* display relative D-efficiency */
          beta=zero)                       /* assumed beta vector, Ho: b=0  */

proc print data=bestcov label;
   id __label;
   label __label = '00'x;                  /* hex null suppress label header*/
   var BrandBranolicious -- CountSix_Bars;
   format _numeric_ zer5.2;
   run;
```

[*]This option is first available with SAS 9.2. It will not be recognized, and it will cause an error in earlier SAS releases.

Some of the results are as follows:

```
                              Cereal Bars
                            Evaluate Design

                             Final Results

                       Design                   1
                       Choice Sets             16
                       Alternatives             4
                       Parameters               7
                       Maximum Parameters      48
                       D-Efficiency       11.5168
                       Relative D-Eff     71.9802
                       D-Error             0.0868
                       1 / Choice Sets     0.0625

                              Cereal Bars
                            Evaluate Design

                                                                    Standard
   n    Variable Name        Label                 Variance    DF      Error

   1    BrandBranolicious    Brand Branolicious     0.06250     1     0.25000
   2    BrandBrantasia       Brand Brantasia        0.06250     1     0.25000
   3    BrandBrantopia       Brand Brantopia        0.06250     1     0.25000
   4    Price_2_89           Price $2.89            0.11111     1     0.33333
   5    Price_2_99           Price $2.99            0.11111     1     0.33333
   6    Price_3_09           Price $3.09            0.11111     1     0.33333
   7    CountSix_Bars        Count Six Bars         0.11111     1     0.33333
                                                               ==
                                                                7

                              Cereal Bars
                            Evaluate Design

                          Brand       Brand     Brand   Price Price Price  Count
                    Branolicious Brantasia Brantopia $2.89 $2.99 $3.09 Six Bars

 Brand Branolicious     0.06         0         0        0     0     0      0
 Brand Brantasia        0            0.06      0        0     0     0      0
 Brand Brantopia        0            0         0.06     0     0     0      0
 Price $2.89            0            0         0        0.11  0     0      0
 Price $2.99            0            0         0        0     0.11  0      0
 Price $3.09            0            0         0        0     0     0.11   0
 Count Six Bars         0            0         0        0     0     0      0.11
```

With the proper coding, the covariances are all zero. The variances for the brand effects are the inverse of the number of choice sets (1/16 = 0.06250).* The four-level brand attribute works perfectly with four-alternative choice sets. The variances for the other parameters are larger. Relative *D*-efficiency is 71.9802. Relative *D*-efficiency is based on a 0 to 100 scale. Note, however, that this relative *D*-efficiency of 71.9802 is a pessimistic statement of the goodness of this design, since *D*-efficiency is measured relative to a hypothetical optimal design that does not have the constraint of a constant alternative. The variances of the parameter estimates are more important when there is a constant alternative than the measure of relative *D*-efficiency.

The preceding steps all used a main-effects model. Alternatively, you could fit separate price and count effects for each brand. These are alternative-specific effects and consist of brand by attribute interactions. The following steps provide an example:

```
title2 'Evaluate Design for Alternative-Specific Model';

%choiceff(data=sasuser.cerealdes,          /* candidate choice sets         */
          init=sasuser.cerealdes(keep=set),/* select these sets from cands  */
          intiter=0,                       /* eval without internal iters   */
                                           /* alternative-specific model    */
                                           /* stdzd orthogonal coding       */
          model=class(brand brand*price brand*count / sta) /
                cprefix=0                  /* lpr=0 labels from just levels */
                lprefix=0,                 /* cpr=0 names from just levels  */
          nalts=4,                         /* number of alternatives        */
          nsets=16,                        /* number of choice sets         */
          options=relative,                /* display relative D-efficiency */
          beta=zero)                       /* assumed beta vector, Ho: b=0  */
```

Now, brand effects are requested as well as brand by price and brand by count interactions. The `cprefix=0` option is specified so that variable names are constructed just from the attribute levels using zero characters of the attribute (or class) variable names. Similarly, the `lprefix=0` option is specified so that variable labels are constructed just from the attribute levels using zero characters of the attribute (or class) variable names or labels. This is because we do not need to see names such as "Brand" or "Price" in our names and labels to understand them. The following results contain the last two output tables, which are what we are most interested in seeing:

*This comparison is only valid when the standardized orthogonal contrast coding is used.

```
                              Cereal Bars
               Evaluate Design for Alternative-Specific Model

                             Final Results

                       Design                  1
                       Choice Sets            16
                       Alternatives            4
                       Parameters             15
                       Maximum Parameters     48
                       D-Efficiency       8.7825
                       Relative D-Eff    54.8908
                       D-Error            0.1139
                       1 / Choice Sets    0.0625

                              Cereal Bars
               Evaluate Design for Alternative-Specific Model

                                                                         Standard
 n   Variable Name          Label                       Variance   DF      Error

 1   Branolicious           Branolicious                 0.06250    1    0.25000
 2   Brantasia              Brantasia                    0.06250    1    0.25000
 3   Brantopia              Brantopia                    0.06250    1    0.25000

 4   Branolicious_2_89      Branolicious * $2.89         0.25000    1    0.50000
 5   Branolicious_2_99      Branolicious * $2.99         0.25000    1    0.50000
 6   Branolicious_3_09      Branolicious * $3.09         0.25000    1    0.50000

 7   Brantasia_2_89         Brantasia * $2.89            0.13889    1    0.37268
 8   Brantasia_2_99         Brantasia * $2.99            0.13889    1    0.37268
 9   Brantasia_3_09         Brantasia * $3.09            0.13889    1    0.37268

10   Brantopia_2_89         Brantopia * $2.89            0.11111    1    0.33333
11   Brantopia_2_99         Brantopia * $2.99            0.11111    1    0.33333
12   Brantopia_3_09         Brantopia * $3.09            0.11111    1    0.33333

13   BranoliciousSix_Bars   Branolicious * Six Bars      0.25000    1    0.50000

14   BrantasiaSix_Bars      Brantasia * Six Bars         0.13889    1    0.37268

15   BrantopiaSix_Bars      Brantopia * Six Bars         0.11111    1    0.33333
                                                                   ==
                                                                   15
```

Now, there are 15 parameters as opposed to the 7 we had previously. There are $(4-1) = 3$ for brand, $(4-1) \times (4-1) = 9$ for the alternative-specific price effects and $(4-1) \times (2-1) = 3$ for the alternative-specific count effects. With 16 choice sets and 4 alternatives, we can estimate at most $16 \times (4-1) = 48$ parameters, so we are still nowhere close to trying to estimate the maximum number of parameters. This design looks good for an alternative-specific effects model. All parameters are estimable, and the variances look reasonable (not overly large relative to 1/16).

You can use the following steps to display the covariance matrix, which is large, in a series of panels:

```
%macro printcov(vars);
   proc print data=bestcov label;
      id __label;
      label __label = '00'x;
      var &vars;
      format _numeric_ zer5.2;
      run;
   %mend;

%printcov(Branolicious Brantasia Brantopia)
%printcov(Branolicious_2_89 Branolicious_2_99 Branolicious_3_09)
%printcov(Brantasia_2_89 Brantasia_2_99 Brantasia_3_09)
%printcov(Brantopia_2_89 Brantopia_2_99 Brantopia_3_09)
%printcov(BranoliciousSix_Bars BrantasiaSix_Bars BrantopiaSix_Bars)
```

The results are as follows:

```
                              Cereal Bars
                 Evaluate Design for Alternative-Specific Model

                                  Branolicious    Brantasia    Brantopia

          Branolicious                0.06           0            0
          Brantasia                   0              0.06         0
          Brantopia                   0              0            0.06
          Branolicious * $2.89        0              0            0
          Branolicious * $2.99        0              0            0
          Branolicious * $3.09        0              0            0
          Brantasia * $2.89           0              0            0
          Brantasia * $2.99           0              0            0
          Brantasia * $3.09           0              0            0
          Brantopia * $2.89           0              0            0
          Brantopia * $2.99           0              0            0
          Brantopia * $3.09           0              0            0
          Branolicious * Six Bars     0              0            0
          Brantasia * Six Bars        0              0            0
          Brantopia * Six Bars        0              0            0
```

```
                              Cereal Bars
                 Evaluate Design for Alternative-Specific Model

                            Branolicious    Branolicious    Branolicious
                              * $2.89         * $2.99         * $3.09

Branolicious                   0               0               0
Brantasia                      0               0               0
Brantopia                      0               0               0
Branolicious * $2.89           0.25            0               0
Branolicious * $2.99           0               0.25            0
Branolicious * $3.09           0               0               0.25
Brantasia * $2.89              0.10            0               0
Brantasia * $2.99              0               0.10            0
Brantasia * $3.09              0               0               0.10
Brantopia * $2.89              0.07            0               0
Brantopia * $2.99              0               0.07            0
Brantopia * $3.09              0               0               0.07
Branolicious * Six Bars        0               0               0
Brantasia * Six Bars           0               0               0
Brantopia * Six Bars           0               0               0


                              Cereal Bars
                 Evaluate Design for Alternative-Specific Model

                                Brantasia    Brantasia    Brantasia
                                 * $2.89      * $2.99      * $3.09

    Branolicious                   0            0            0
    Brantasia                      0            0            0
    Brantopia                      0            0            0
    Branolicious * $2.89           0.10         0            0
    Branolicious * $2.99           0            0.10         0
    Branolicious * $3.09           0            0            0.10
    Brantasia * $2.89              0.14         0            0
    Brantasia * $2.99              0            0.14         0
    Brantasia * $3.09              0            0            0.14
    Brantopia * $2.89              0.04         0            0
    Brantopia * $2.99              0            0.04         0
    Brantopia * $3.09              0            0            0.04
    Branolicious * Six Bars        0            0            0
    Brantasia * Six Bars           0            0            0
    Brantopia * Six Bars           0            0            0
```

```
                            Cereal Bars
              Evaluate Design for Alternative-Specific Model

                              Brantopia    Brantopia    Brantopia
                               * $2.89      * $2.99      * $3.09

     Branolicious                0            0            0
     Brantasia                   0            0            0
     Brantopia                   0            0            0
     Branolicious * $2.89        0.07         0            0
     Branolicious * $2.99        0            0.07         0
     Branolicious * $3.09        0            0            0.07
     Brantasia * $2.89           0.04         0            0
     Brantasia * $2.99           0            0.04         0
     Brantasia * $3.09           0            0            0.04
     Brantopia * $2.89           0.11         0            0
     Brantopia * $2.99           0            0.11         0
     Brantopia * $3.09           0            0            0.11
     Branolicious * Six Bars     0            0            0
     Brantasia * Six Bars        0            0            0
     Brantopia * Six Bars        0            0            0


                            Cereal Bars
              Evaluate Design for Alternative-Specific Model

                           Branolicious    Brantasia     Brantopia
                            * Six Bars     * Six Bars    * Six Bars

   Branolicious                0              0             0
   Brantasia                   0              0             0
   Brantopia                   0              0             0
   Branolicious * $2.89        0              0             0
   Branolicious * $2.99        0              0             0
   Branolicious * $3.09        0              0             0
   Brantasia * $2.89           0              0             0
   Brantasia * $2.99           0              0             0
   Brantasia * $3.09           0              0             0
   Brantopia * $2.89           0              0             0
   Brantopia * $2.99           0              0             0
   Brantopia * $3.09           0              0             0
   Branolicious * Six Bars     0.25           0.10          0.07
   Brantasia * Six Bars        0.10           0.14          0.04
   Brantopia * Six Bars        0.07           0.04          0.11
```

There are some nonzero but small covariances off the diagonal.

There is one more test that could be run before a design is used. The `%MktDups` macro in the following step checks the design to see if any choice sets are duplicates of any other choice sets:

```
%mktdups(branded,                         /* a design with brands          */
        data=sasuser.cerealdes,           /* the input design to evaluate  */
        factors=brand price count,        /* factors in the design         */
        nalts=4)                          /* number of alternatives        */
```

The first parameter is a positional parameter. There is no *key-word=* preceding its value, and it must always be specified. We specify that this is a branded design as opposed to a generic design (bundles of attributes with no brands). We also specify the input SAS data set, the factors (attributes) in the design, and the number of alternatives. The results are as follows:

```
Design:          Branded
Factors:         brand price count
                 Brand
                 Count Price
Duplicate Sets:  0
```

The first line of the table tells us that this is a branded design. The second line tells us the factors as specified in the `factors=` option. These are followed by the actual variable names for the factors. The last line reports the number of duplicates. In this case, there are no duplicate choice sets. (Of course, we already knew that from the *n*-way frequencies in the `%MktEval` output.) If there had been duplicate choice sets, then changing the random number seed will sometimes help. Sometimes, changing other aspects of the design or the approach for making the design will help.

Next, the questionnaire is designed. Two sample choice sets are as follows:

Branolicious	Brantopia	Brantasia	No Purchase
$2.89	$2.99	$2.99	
Six Bars	Six Bars	Six Bars	

Branolicious	Brantopia	Brantasia	No Purchase
$3.19	$2.99	$3.19	
Six Bars	Eight Bars	Eight Bars	

In practice, data collection is usually much more elaborate than this. It might involve art work or photographs, and the choice sets might be presented and the data might be collected through personal interview or over the Web. However the choice sets are presented and the data are collected, the essential ingredients remain the same. Subjects are shown sets of alternatives and are asked to make a choice, then they go on to the next set. Each subject sees all 16 choice sets and chooses one alternative from each. The data for each subject consist of 16 integers in the range 1 to 4 showing which alternative was chosen.

The data are collected and entered into a SAS data set as follows:

```
title2 'Read Data';

data results;
   input Subject (r1-r16) (1.);
   datalines;
  1 1331132331312213
  2 3231322131312233
  3 1233332111132233
  4 1211232111313233
  5 1233122111312233
  6 3231323131212313
  7 3231232131332333
  8 3233332131322233
  9 1223332111333233
 10 1332132111233233
 11 1233222211312333
 12 1221332111213233
 13 1231332131133233
 14 3211333211313233
 15 3313332111122233
 16 3321123231331223
 17 3223332231312233
 18 3211223311112233
 19 1232332111132233
 20 1213233111312413
 21 1333232131212233
 22 3321322111122231
 23 3231122131312133
 24 1232132111311333
 25 3113332431213233
 26 3213132141331233
 27 3221132111312233
 28 3222333131313231
 29 1221332131312231
 30 3233332111212233
 31 1221332111342233
 32 2233232111111211
 33 2332332131211231
 34 2221132211312411
 35 1232233111332233
 36 1231333131322333
 37 1231332111331333
 38 1223132211233331
 39 1321232131211231
 40 1223132331321233
;
```

There is one row for each subject containing the number of the chosen alternatives for each of the 16 choice sets.

The `%MktMerge` macro in the following step merges the data and the design and creates the dependent variable:

```
title2 'Merge Data and Design';

%mktmerge(design=sasuser.cerealdes, /* input design                         */
          data=results,             /* input data set                       */
          out=res2,                 /* output data set with design and data */
          nsets=16,                 /* number of choice sets                */
          nalts=4,                  /* number of alternatives               */
          setvars=r1-r16)           /* variables with the chosen alt nums   */
```

The `design=` input data set has one row for each alternative of each choice set. The `data=` input data set has one row for each subject. The `out=` data set has one row for each alternative of each choice set for each subject (in this case, there are $4 \times 16 \times 40 = 2560$ rows). The following step displays the first four choice sets for the first subject:

```
title2 'Design and Data Both';

proc print data=res2(obs=16);
   by set subject;   id set subject;
   run;
```

The first four choice sets for the first subject are as follows:

```
                              Cereal Bars
                          Design and Data Both

          Set    Subject    Brand          Price      Count       c

           1        1       Branolicious   $2.89    Six Bars      1
                            Brantopia      $2.99    Six Bars      2
                            Brantasia      $2.99    Six Bars      2
                            None                                  2

           2        1       Branolicious   $3.19    Six Bars      2
                            Brantopia      $2.99    Eight Bars    2
                            Brantasia      $3.19    Eight Bars    1
                            None                                  2

           3        1       Branolicious   $3.09    Eight Bars    2
                            Brantopia      $2.99    Six Bars      2
                            Brantasia      $3.09    Eight Bars    1
                            None                                  2
```

```
                4        1       Branolicious    $3.09    Six Bars      1
                                 Brantopia       $3.19    Six Bars      2
                                 Brantasia       $3.19    Six Bars      2
                                 None                                   2
```

The dependent variable is `c`. A 1 in `c` indicates first choice, and a 2 indicates the alternatives that were not chosen (second or subsequent choices).

This following step codes the design for analysis:

```
title2 'Code the Independent Variables';

proc transreg design norestoremissing data=res2;
   model class(brand price count);
   id subject set c;
   output out=coded(drop=_type_ _name_ intercept) lprefix=0;
   run;
```

We will typically use PROC TRANSREG for coding because it has a series of options that are useful for coding choice models. This step does not do any analysis; it just codes. This is because the `design` option specifies that only coding is to be done. **The `norestoremissing` option creates zeros in the indicator variables for `class` variables with missing values instead of by default replacing the zeros with missings.** You will need to use the `norestoremissing` option whenever there is a constant or None alternative that is indicated in whole or in part by missing values. The `data=` option names the design to code. The `model` statement names the product attributes. Since no options are specified in the `class` specification, the default reference-cell coding is used.[*] The `id` statement names the other variables that we will need for analysis. The `output` statement creates and `out=coded` data set with the coded design, drops a few variables that we do not need, and uses the `lprefix=0` option to get labels for the parameters from just the levels and not the input variable names and labels.

The following steps display the coded results for the first subject for the first four choice sets:

```
proc print data=coded(obs=16) label;
   title3 'ID Information and the Dependent Variable';
   format price price. count count.;
   var Brand Price Count Subject Set c;
   by set subject;   id set subject;
   run;

proc print data=coded(obs=16) label;
   title3 'ID Information and the Coding of Brand';
   format price price. count count.;
   var brandbranolicious brandbrantasia brandbrantopia brand;
   by set subject;   id set subject;
   run;
```

[*]Note that there is no problem with using the standardized orthogonal contrast coding to make the design and using reference cell, effects, or any other coding when you use the design. *D*-efficiency guarantees that you will get equivalent results if you change codings.

```
proc print data=coded(obs=16) label;
   title3 'ID Information and the Coding of Price and Count';
   format price price. count count.;
   var Price_2_89 Price_2_99 Price_3_09 CountSix_Bars Price Count;
   by set subject;   id set subject;
   run;
```

The coded design for the first four choice sets is shown in the following three panels:

```
                              Cereal Bars
                     Code the Independent Variables
               ID Information and the Dependent Variable

 Set    Subject    Brand           Price      Count       Subject    Set    c

  1        1       Branolicious    $2.89    Six Bars         1        1     1
                   Brantopia       $2.99    Six Bars         1        1     2
                   Brantasia       $2.99    Six Bars         1        1     2
                   None                                      1        1     2

  2        1       Branolicious    $3.19    Six Bars         1        2     2
                   Brantopia       $2.99    Eight Bars       1        2     2
                   Brantasia       $3.19    Eight Bars       1        2     1
                   None                                      1        2     2

  3        1       Branolicious    $3.09    Eight Bars       1        3     2
                   Brantopia       $2.99    Six Bars         1        3     2
                   Brantasia       $3.09    Eight Bars       1        3     1
                   None                                      1        3     2

  4        1       Branolicious    $3.09    Six Bars         1        4     1
                   Brantopia       $3.19    Six Bars         1        4     2
                   Brantasia       $3.19    Six Bars         1        4     2
                   None                                      1        4     2

                              Cereal Bars
                     Code the Independent Variables
                 ID Information and the Coding of Brand

   Set    Subject    Branolicious    Brantasia    Brantopia    Brand

    1        1             1             0            0        Branolicious
                           0             0            1        Brantopia
                           0             1            0        Brantasia
                           0             0            0        None

    2        1             1             0            0        Branolicious
                           0             0            1        Brantopia
                           0             1            0        Brantasia
                           0             0            0        None
```

```
         3        1             1             0            0        Branolicious
                                0             0            1        Brantopia
                                0             1            0        Brantasia
                                0             0            0        None

         4        1             1             0            0        Branolicious
                                0             0            1        Brantopia
                                0             1            0        Brantasia
                                0             0            0        None

                                 Cereal Bars
                        Code the Independent Variables
               ID Information and the Coding of Price and Count

                                                      Six
        Set    Subject    $2.89    $2.99    $3.09    Bars    Price      Count

         1        1         1        0        0        1     $2.89    Six Bars
                            0        1        0        1     $2.99    Six Bars
                            0        1        0        1     $2.99    Six Bars
                            0        0        0        0

         2        1         0        0        0        1     $3.19    Six Bars
                            0        1        0        0     $2.99    Eight Bars
                            0        0        0        0     $3.19    Eight Bars
                            0        0        0        0

         3        1         0        0        1        0     $3.09    Eight Bars
                            0        1        0        1     $2.99    Six Bars
                            0        0        1        0     $3.09    Eight Bars
                            0        0        0        0

         4        1         0        0        1        1     $3.09    Six Bars
                            0        0        0        1     $3.19    Six Bars
                            0        0        0        1     $3.19    Six Bars
                            0        0        0        0
```

The following steps fit the choice model:

```
%phchoice(on)                      /* customize PHREG for a choice model   */

title2 'Multinomial Logit Discrete Choice Model';

proc phreg data=coded brief;
   model c*c(2) = &_trgind / ties=breslow;
   strata subject set;
   run;

%phchoice(off)                     /* restore PHREG to a survival PROC     */
```

Notice that we use the `%PHChoice` macro to customize the output from PROC PHREG so that it looks more like discrete choice output and less like survival analysis output. The choice model is a special case of a survival-analysis model. The `brief` option is used to get a brief summary of the pattern of chosen and not chosen alternatives. This is very useful for checking data entry. Before the model equals sign, the first mention of `c` indicates the chosen alternative and the second mention of `c` indicates the alternatives that were not chosen. The list in parentheses indicates that values of 2 or greater were not chosen. When we set `c` as we did in this example (1 means first choice and 2 means unobserved second or subsequent choices), we will always specify `c*c(2)` before the equal sign as our response specification. A macro variable that PROC TRANSREG creates is specified after the equal sign. This macro variable is always called `&_trgind` and it contains the list of coded variables. The list of variables PROC TRANSREG creates will change for every study, but you can always use the macro variable `&_trgind` to get the list. The `ties=breslow` option specifies the likelihood function that we want for the multinomial logit discrete choice model. Each subject and set combination makes a contribution to the likelihood function, so those variables are specified in the `strata` statement. The results are as follows:

```
                                Cereal Bars
                    Multinomial Logit Discrete Choice Model

                           The PHREG Procedure

                            Model Information

                  Data Set                 WORK.CODED
                  Dependent Variable       c
                  Censoring Variable       c
                  Censoring Value(s)       2
                  Ties Handling            BRESLOW

                Number of Observations Read        2560
                Number of Observations Used        2560

     Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                Number of       Number of          Chosen          Not
     Pattern      Choices    Alternatives    Alternatives       Chosen

           1          640               4               1            3

                            Convergence Status

              Convergence criterion (GCONV=1E-8) satisfied.
```

```
                         Model Fit Statistics

                              Without           With
                Criterion     Covariates     Covariates

                -2 LOG L        1774.457       1142.630
                AIC             1774.457       1156.630
                SBC             1774.457       1187.860

                Testing Global Null Hypothesis: BETA=0

        Test                 Chi-Square       DF     Pr > ChiSq

        Likelihood Ratio       631.8271        7         <.0001
        Score                  518.1014        7         <.0001
        Wald                   275.0965        7         <.0001

                              Cereal Bars
                Multinomial Logit Discrete Choice Model

                          The PHREG Procedure

                 Multinomial Logit Parameter Estimates

                        Parameter      Standard
                  DF     Estimate         Error    Chi-Square    Pr > ChiSq

Branolicious       1      2.64506       0.47268       31.3142        <.0001
Brantasia          1      2.94600       0.47200       38.9571        <.0001
Brantopia          1      2.44876       0.47416       26.6706        <.0001
$2.89              1      2.69907       0.20307      176.6557        <.0001
$2.99              1      1.72036       0.17746       93.9845        <.0001
$3.09              1      0.76407       0.17437       19.2008        <.0001
Six Bars           1     -0.54645       0.11899       21.0912        <.0001
```

Notice near the top of the output that there was one pattern of results. There were 640 times (16 choice sets times 40 people) that four alternatives were presented and one was chosen. This table, which was produced by the `brief` option, provides a check on the data entry. Usually, the number of alternatives is the same in all choice sets, as it is here. Multiple patterns would mean a data entry error had occurred. The "Multinomial Logit Parameter Estimates" table is of primary interest. All of the part-worth utilities (parameter estimates) are significant, and the clearest pattern in the results is that the lower prices have the highest utility (the larger parameter estimates).

The following steps are not necessary, but they show some of the details about how the parameters are interpreted and how they can be used to find the utility of each combination. Recall that the choice model has the following form

$$p(c_i|C) = \frac{\exp(U(c_i))}{\sum_{j=1}^{m} \exp(U(c_j))} = \frac{\exp(\mathbf{x}_i\boldsymbol{\beta})}{\sum_{j=1}^{m} \exp(\mathbf{x}_j\boldsymbol{\beta})}$$

The following steps create the predicted utility of each alternative, $\widehat{U(c_i)}$, from each alternative's attributes, $\mathbf{x}_j$, and the vector of parameter estimates, $\hat{\boldsymbol{\beta}}$ (and additionally the element-wise products of $\mathbf{x}_j$, and $\hat{\boldsymbol{\beta}}$):

```
proc sort data=coded nodupkeys out=combos(drop=subject -- c);
   by brand price count;
   run;

data utils(drop=i);
   set combos;
   array b[7] _temporary_ (2.7 2.3 2.9 2.9 1.7 0.7 -1.2);
   array x[7] brandbranolicious -- countsix_bars;
   u = 0;
   do i = 1 to 7;
      x[i] = b[i] * x[i];
      u + x[i];
      end;
   run;

proc print label noobs split='-';
   title2 'Part-Worth Utility Report';
   label BrandBranolicious = 'Bran-olic-ious-'
         BrandBrantasia    = 'Bran-tas -ia  -'
         BrandBrantopia    = 'Bran-top -ia  -';
   id u;
   run;
```

The first step sorts the coded design data set by brand, price, and count, the three attributes, and deletes all duplicates. This creates 25 combinations: 3 brands times 4 prices times 2 counts plus one constant alternative. The DATA step multiplies each indicator variable in the resulting data set by its associated parameter estimate or part-worth utility. It also sums the appropriate part-worth utilities across all of the attributes and stores the result in the variable u. The final step displays the results, which are as follows:

```
                              Cereal Bars
                        Part-Worth Utility Report

       Bran Bran Bran
       olic tas  top                         Six
   u   ious ia   ia   $2.89 $2.99 $3.09 Bars Brand        Price   Count

  4.4  2.7  0.0  0.0  2.9   0.0   0.0  -1.2 Branolicious $2.89 Six Bars
  5.6  2.7  0.0  0.0  2.9   0.0   0.0   0.0 Branolicious $2.89 Eight Bars
  3.2  2.7  0.0  0.0  0.0   1.7   0.0  -1.2 Branolicious $2.99 Six Bars
  4.4  2.7  0.0  0.0  0.0   1.7   0.0   0.0 Branolicious $2.99 Eight Bars
  2.2  2.7  0.0  0.0  0.0   0.0   0.7  -1.2 Branolicious $3.09 Six Bars
  3.4  2.7  0.0  0.0  0.0   0.0   0.7   0.0 Branolicious $3.09 Eight Bars
```

```
1.5  2.7  0.0  0.0  0.0   0.0   0.0  -1.2 Branolicious $3.19 Six Bars
2.7  2.7  0.0  0.0  0.0   0.0   0.0   0.0 Branolicious $3.19 Eight Bars
4.0  0.0  2.3  0.0  2.9   0.0   0.0  -1.2 Brantasia    $2.89 Six Bars
5.2  0.0  2.3  0.0  2.9   0.0   0.0   0.0 Brantasia    $2.89 Eight Bars
2.8  0.0  2.3  0.0  0.0   1.7   0.0  -1.2 Brantasia    $2.99 Six Bars
4.0  0.0  2.3  0.0  0.0   1.7   0.0   0.0 Brantasia    $2.99 Eight Bars
1.8  0.0  2.3  0.0  0.0   0.0   0.7  -1.2 Brantasia    $3.09 Six Bars
3.0  0.0  2.3  0.0  0.0   0.0   0.7   0.0 Brantasia    $3.09 Eight Bars
1.1  0.0  2.3  0.0  0.0   0.0   0.0  -1.2 Brantasia    $3.19 Six Bars
2.3  0.0  2.3  0.0  0.0   0.0   0.0   0.0 Brantasia    $3.19 Eight Bars
4.6  0.0  0.0  2.9  2.9   0.0   0.0  -1.2 Brantopia    $2.89 Six Bars
5.8  0.0  0.0  2.9  2.9   0.0   0.0   0.0 Brantopia    $2.89 Eight Bars
3.4  0.0  0.0  2.9  0.0   1.7   0.0  -1.2 Brantopia    $2.99 Six Bars
4.6  0.0  0.0  2.9  0.0   1.7   0.0   0.0 Brantopia    $2.99 Eight Bars
2.4  0.0  0.0  2.9  0.0   0.0   0.7  -1.2 Brantopia    $3.09 Six Bars
3.6  0.0  0.0  2.9  0.0   0.0   0.7   0.0 Brantopia    $3.09 Eight Bars
1.7  0.0  0.0  2.9  0.0   0.0   0.0  -1.2 Brantopia    $3.19 Six Bars
2.9  0.0  0.0  2.9  0.0   0.0   0.0   0.0 Brantopia    $3.19 Eight Bars
0.0  0.0  0.0  0.0  0.0   0.0   0.0   0.0 None             .          .
```

At the bottom of the output, you can see that the utility for the constant alternative is 0, and the coded variables are such that it cannot possibly be anything else. The utilities for all of the other alternatives are computed relative to the constant. If the constant alternative were the most preferred, all of the other utilities would be negative. In this case, the constant alternative is least preferred and the other utilities are positive. Each of the 25 different combinations has a different pattern of indicator variables and parameters. “None” is the reference level for brand, and it has a zero part-worth utility for brand. The other brands have positive part-worth utilities relative to “None”. The reference level for price is $3.19, and it has a zero part-worth utility. The other prices have positive part-worth utilities relative to $3.19 that increase as price decreases. Eight bars is the reference level for count, and it has a zero part-worth utility. The other count (6 bars) has negative part-worth utility relative to 8 bars. Other coding schemes would produce different but equivalent patterns of results.

In summary, this example shows the basic steps in designing, processing, and analyzing a choice experiment using the approach that creates the design directly using the `%MktEx` macro. The remaining examples illustrate other approaches that are also commonly used. Also, pages 285 through 663 have many more examples, much greater detail, and show how to use other tools.

Example 2: The Linear Arrangement Approach with Restrictions

In this example, we create a design for the same study as in the previous example. The product line is the same three fictitious breakfast bars with the same attributes as before. However, this time, rather than making a design in which all of the attributes of all of the alternatives are balanced and orthogonal, we impose restrictions on the design. This is not a full example, rather it simply shows how to modify the design-creation steps in the first example to impose restrictions. We will restrict the design to avoid choice sets where attributes are constant. That is, we want to select just the choice sets where neither price nor count is constant within a choice set. We use the `%MktEx` macro as follows to get a restricted factorial design for this problem as follows:

```
%macro res;
   if x1 = x3 & x1 = x5 then bad = 1;
   if x2 = x4 & x2 = x6 then bad = bad + 1;
   %mend;

%mktex(4 2  4 2  4 2,       /* factor level list for all attrs and alts     */
       n=16,                /* number of choice sets                        */
       restrictions=res,    /* name of the restrictions macro               */
       options=resrep,      /* detailed report on restrictions              */
       seed=17)             /* random number seed                           */
```

Restrictions are written with PROC IML statements embedded in a macro. You provide the `%MktEx` macro with the name of the restrictions macro by using the `restrictions=`*macro-name* option. In the restrictions macro, you compute an IML scalar called `bad` that quantifies the badness of the design. In this case, since the restrictions are entirely within choice set, you can just quantify the badness of one choice set at at time by evaluating the values in the scalars `x1-x6`, which correspond to the six attributes. If it is easier to use indexing to write the restrictions, you can instead use the vector `x`, where `x[1]` = `x1`, ..., `x[6]` = `x6`. Furthermore, while it is not illustrated in this example, the levels of the attributes for the entire design are stored in a matrix called `xmat`, and you can use these values as well as the values in `x` to impose restrictions across choice sets. The scalar `bad` is automatically initialized to zero by the `%MktEx` macro. In this example, `bad` is set to 1 if the `Price` variable (which is made from `x1`, `x3`, and `x5`) is constant. The scalar `bad` is incremented by 1 if the `Count` variable (which is made from `x2`, `x4`, and `x6`) is constant. The `if` statements use the Boolean syntax of PROC IML. We must use the following IML logical operators, which do not have all of the same syntactical alternatives as DATA step operators:

Specify	For	Do Not Specify
=	equals	EQ
∧ = or ¬ =	not equals	NE
<	less than	LT
<=	less than or equal to	LE
>	greater than	GT
>=	greater than or equal to	GE
&	and	AND
|	or	OR
∧ or ¬	not	NOT
`a` <= `b` & `b` <= `c`	range check	`a` <= `b` <= `c`

Your macro can look at several scalars, along with a vector and a matrix in quantifying badness, and it must store its results in `bad`. The following names are available:

`i` – is a scalar that contains the number of the row currently being changed or evaluated.
If you are writing restrictions that use the variable `i`, you almost certainly should specify `options=nosort`.

`try` – is a scalar similar to `i`, which contains the number of the row currently being changed. However, `try`, starts at zero and is incremented for each row, but it is only set back to zero when a new design starts, not when `%MktEx` reaches the last row. Use `i` as a matrix index and `try` to evaluate how far `%MktEx` is into the process of constructing the design.

`x` – is a row vector of factor levels for row `i` that always contains integer values beginning with 1 and continuing on to the number of levels for each factor. These values are always one-based, even if `levels=` is specified.

`x1` is the same as `x[1]`, `x2` is the same as `x[2]`, and so on.

`j1` – is a scalar that contains the number of the column currently being changed. In the steps where the badness macro is called once per row, `j1` = 1.

`j2` – is a scalar that contains the number of the other column currently being changed (along with `j1`) with `exchange=2`. Both `j1` and `j2` are defined when the `exchange=` value is greater than or equal to two. This scalar will not exist with `exchange=1`. In the steps where the badness macro is called once per row, `j1` = `j21` = 1.

`j3` – is a scalar that contains the number of the third column currently being changed (along with `j1` and `j2`) with `exchange=3` and larger `exchange=` values. This scalar will not exist with `exchange=1` and `exchange=2`. If and only if the `exchange=`value is greater than 3, there will be a `j4` and so on. In the steps where the badness macro is called once per row, `j1` = `j2` = `j3` = 1.

`xmat` – is the entire `x` matrix. Note that the *ith* row of `xmat` is often different from `x` since `x` contains information about the exchanges being considered, whereas `xmat` contains the current design.

`bad` – results: 0 – fine, or the number of violations of restrictions. You can make this value large or small, and you can use integers or real numbers. However, the values should always be nonnegative. When there are multiple sources of design badness, it is sometimes good to scale the different sources on different scales so that they do not trade off against each other. For example, for the first source, you might multiply the number of violations by 1000, by 100 for another source, by 10 for another source, by 1 for another source, and even sometimes by 0.1 or 0.01 for another source. The final badness is the sum of `bad`, `__pbad` (when it exists), and `__bbad` (when it exists). The scalars `__pbad` and `__bbad` are explained next.

`__pbad` – is the badness from the `partial=` option. When `partial=` is not specified, this scalar does not exist. Your macro can weight this value, typically by multiplying it times a constant, to differentially weight the contributors to badness, e.g.: `__pbad = __pbad * 10`.

`__bbad` – is the badness from the `balance=` option. When `balance=` is not specified, this scalar does not exist. Your macro can weight this value, typically by multiplying it times a constant, to differentially weight the contributors to badness, e.g.: `__bbad = __bbad * 100`.

Do not use these names (other than `bad`) for intermediate values!

Other than that, you can create intermediate variables without worrying about conflicts with the names in the macro. The levels of the factors for one row of the experimental design are stored in a vector `x`, and the first level is always 1, the second always 2, and so on. All restrictions must be defined in terms of `x[j]` (or alternatively, `x1`, `x2`, ..., and perhaps the other matrices).

One other option is specified in this example, and that is `options=resrep`. It is a good idea to specify this option with restrictions until you are sure that your restrictions macro is correct. It provides detailed information about the swaps that the `%MktEx` macro performs and its success in imposing restrictions.

Part of the iteration history table, with a lot of `options=resrep` information and other information deleted, is as follows:

```
                              Cereal Bars

                        Algorithm Search History

                                Current          Best
Design    Row,Col  D-Efficiency  D-Efficiency  Notes
----------------------------------------------------------
     1      Start       91.5504                Can
     1      1           91.5504                0 Violations
     1      2           91.5504                0 Violations
     .
     .
     .
     1     16           91.5504                0 Violations
     1      1           91.5504                0 Violations
     1      2   1       91.5504       91.5504  Conforms
     1        End       91.5504

     2      Start      100.0000                Tab
     2      1           95.2509                0 Violations
     2      2           95.2509                0 Violations
     .
     .
     .
     2     14   1       87.8163                Conforms
     2        End       89.5326
     .
     .
     .

    12      Start       61.3114                Ran,Mut,Ann
    12      1           61.3115                0 Violations
    12      2           65.3098                0 Violations
     .
     .
     .
    12     16   1       79.4843                Conforms
    12        End       89.1152
     .
     .
     .
```

```
          21      Start       53.6255                 Ran,Mut,Ann
          21      1           60.0902                 0 Violations
          21      2           60.0902                 0 Violations
           .
           .
           .
          21     16   1       81.3052                 Conforms
          21        End       89.5758

NOTE: Performing 1000 searches of 360 candidates.


                            Cereal Bars

                       Design Search History

                              Current         Best
           Design    Row,Col  D-Efficiency  D-Efficiency  Notes
           ----------------------------------------------------------
                0    Initial       91.5504       91.5504  Ini

                1      Start       91.5504                Can
                1      1           91.5504                0 Violations
                1      2           91.5504                0 Violations
                1      3           91.5504                0 Violations
                .
                .
                .
                1     16           91.5504                0 Violations
                1      1           91.5504                0 Violations
                1      2   1       91.5504       91.5504  Conforms
                1        End       91.5504
```

```
                                 Cereal Bars

                          Design Refinement History

                                  Current          Best
              Design    Row,Col  D-Efficiency  D-Efficiency  Notes
              ----------------------------------------------------------
                   0    Initial       91.5504       91.5504  Ini

                   1      Start       85.9351                Pre,Mut,Ann
                   1      1           85.9351                0 Violations
                   1      2           85.9351                0 Violations
                   1      3           87.0050                0 Violations
                   .
                   .
                   .
                   1     16   1       91.5504       91.5504
                   1      2   1       91.5504       91.5504
                   1        End       91.5504
                   .
                   .
                   .
                   9      Start       91.5504                Pre,Mut,Ann
                   9      1           91.5504                0 Violations
                   9      2           91.5504                0 Violations
                   .
                   .
                   .
                   9     16           91.5504                0 Violations
                   9      1           91.5504                0 Violations
                   9      2   1       91.5504       91.5504  Conforms
                   9        End       87.4793

        NOTE: Stopping since it appears that no improvement is possible.

                            The OPTEX Procedure

                          Class Level Information

                           Class  Levels  Values

                           x1       4     1 2 3 4
                           x2       2     1 2
                           x3       4     1 2 3 4
                           x4       2     1 2
                           x5       4     1 2 3 4
                           x6       2     1 2
```

```
                                                                    Average
                                                                   Prediction
       Design                                                       Standard
       Number    D-Efficiency     A-Efficiency     G-Efficiency       Error
       ------------------------------------------------------------------------
            1         91.5504          80.7068          93.8194          0.9014
```

The iteration history table for the `%MktEx` macro is described in detail in the discrete choice chapter starting on page 285. In this example, just note a few things. The `%MktEx` macro is successful in making the design conform to all restrictions. In all cases, it reports "0 Violations" of the restrictions. In some cases, a design with 100% *D*-efficiency is replaced by a design with a lower *D*-efficiency as the restrictions are imposed. The final *D*-efficiency is 91.5504. Designs with this same *D*-efficiency are repeatedly found, which often indicates that an optimal design was found.

The following steps create the choice design from the linear arrangement and display the results:

```
title2 'Create the Choice Design Key';

data key;
   input
Brand $ 1-12    Price $    Count $; datalines;
Branolicious    x1         x2
Brantopia       x3         x4
Brantasia       x5         x6
None            .          .
;

title2 'Create Choice Design from Linear Arrangement';

proc format;
   value price 1 = $2.89 2 = $2.99 3 = $3.09 4 = $3.19 . = ' ';
   value count 1 = 'Six Bars' 2 = 'Eight Bars'  . = ' ';
   run;

%mktroll(design=randomized,  /* input randomized linear arrangement         */
         key=key,            /* rules for making choice design              */
         alt=brand,          /* brand or alternative label var              */
         out=cerealdes)      /* output choice design                        */

title2;
proc print; format price price. count count.; id set; by set; run;
```

The choice design, with restrictions, is as follows:

```
               Cereal Bars

Set    Brand           Price      Count

  1    Branolicious    $3.09    Six Bars
       Brantopia       $2.89    Six Bars
       Brantasia       $3.09    Eight Bars
       None

  2    Branolicious    $2.89    Six Bars
       Brantopia       $2.99    Eight Bars
       Brantasia       $3.19    Six Bars
       None

  3    Branolicious    $2.99    Eight Bars
       Brantopia       $2.99    Eight Bars
       Brantasia       $3.09    Six Bars
       None

  4    Branolicious    $2.89    Eight Bars
       Brantopia       $2.89    Six Bars
       Brantasia       $2.99    Eight Bars
       None

  5    Branolicious    $3.19    Eight Bars
       Brantopia       $3.19    Six Bars
       Brantasia       $3.09    Six Bars
       None

  6    Branolicious    $3.09    Six Bars
       Brantopia       $3.19    Eight Bars
       Brantasia       $2.99    Eight Bars
       None

  7    Branolicious    $2.89    Six Bars
       Brantopia       $3.09    Eight Bars
       Brantasia       $3.09    Eight Bars
       None

  8    Branolicious    $3.19    Six Bars
       Brantopia       $2.99    Six Bars
       Brantasia       $2.99    Eight Bars
       None

  9    Branolicious    $3.09    Eight Bars
       Brantopia       $2.99    Six Bars
       Brantasia       $2.89    Eight Bars
       None
```

```
10     Branolicious    $3.19    Eight Bars
       Brantopia       $2.89    Eight Bars
       Brantasia       $3.19    Six Bars
       None

11     Branolicious    $2.89    Eight Bars
       Brantopia       $3.19    Six Bars
       Brantasia       $2.89    Six Bars
       None

12     Branolicious    $3.09    Eight Bars
       Brantopia       $3.09    Six Bars
       Brantasia       $2.99    Six Bars
       None

13     Branolicious    $3.19    Six Bars
       Brantopia       $3.09    Eight Bars
       Brantasia       $2.89    Eight Bars
       None

14     Branolicious    $2.99    Six Bars
       Brantopia       $3.19    Six Bars
       Brantasia       $3.19    Eight Bars
       None

15     Branolicious    $2.99    Six Bars
       Brantopia       $2.89    Eight Bars
       Brantasia       $2.89    Six Bars
       None

16     Branolicious    $2.99    Eight Bars
       Brantopia       $3.09    Six Bars
       Brantasia       $3.19    Eight Bars
       None
```

This example provides an illustration of a restrictions macro in a context that is simple enough that it is easy to write the restrictions macro and nothing is likely to go wrong. Many other uses of restrictions are not this simple. The discrete choice chapter starting on page 285 and the `%MktEx` macro documentation starting on page 1017 provide more information about restricting designs with the `%MktEx` macro. There are a few things to always keep in mind when writing restrictions:

- Ensure that you are specifying a set of restrictions that are possible. It is not uncommon for people to write a set of restrictions that cannot possibly be satisfied (for example, `bad = (a >= b) + (b >= a);` although most errors in restriction specifications are not nearly this obvious).
- You can specify restrictions that prohibit certain combinations of attributes from appearing together. You can ask for interactions involving those attributes. However, if you do both, you might find that the *D*-efficiency of your design is zero. Depending on what you specify, you might not be able to have both estimable interactions and restrictions. In order for an interaction parameter to be estimable, certain combinations of levels must appear in the design. If you prohibit

those combinations from occurring, the parameter cannot be estimated.

- Badness must be quantified. When there are multiple sources of badness, badness must be quantified as granularly as you can. If you simply set `bad` to zero when all is fine and nonzero when all is not fine, you might not be giving `%MktEx` enough information to impose restrictions. You have to quantify badness in a way that tells `%MktEx` when it makes a change that is closer to the desired result even when it does not achieve the desired result. Without that information, `%MktEx` does not know when it is moving in the right direction. For example, if your badness criterion forces some attributes to be constant, your badness criterion should reflect how far the varying attributes are from constant when too few attributes are constant.

- When there are multiple independent sources of badness, you might have to differentially weight the sources (e.g. give one part a weight of 1, another a weight of 10, another a weight of 50, and so on). It might not matter which part gets what weight. Often, the important thing is to provide some differential weighting so that sources of badness never trade off against one another. For example, if all sources get an implicit weight of 1, then every time `%MktEx` makes progress on one source it might make the other source worse. Differential weighting can help prevent this.

- Always use `options=resrep` until you are sure you are creating the design correctly. The information that is provided in the iteration history table with `options=resrep` is often very helpful in diagnosing problems with the restrictions macro.

- It is good practice to use `options=quickr` at first with restrictions, unless you have a very small problem like the one in this example. This makes the macro run faster and just tries to make a single design. Use this option to minimize run time until you are sure your restrictions macro is correct, then remove it and let the macro run longer to make the final design. There are many other options to minimize run time for complicated designs while you are checking your code.

- Use the `out=design` data set with restrictions and do not randomize the design. Randomization can destroy the structure imposed by the restrictions.

- Do not change the values of `i`, `try`, `x`, `x1 - x`*n*, `j1`, `j2`, `j3`, `xmat`, `__pbad`, or `__bbad` in your restrictions macro. Do not use these names for any intermediate matrices. Be particularly careful to avoid using `i` as an index in a `do` loop. Use `j` or `k` instead. Do not ignore warnings that you are changing macros in your restrictions macro.

- Restrictions are written in PROC IML, the interactive matrix language, which has a rich variety of matrix and vector operations. Do not use long series of scalar operations when simple vector or matrix operations are available. The following two macros are equivalent, although the first is obviously easier to construct than the second:

```
%macro bad; * The easy way with vectors;
   c1 = sum(x[, 1:32] = 1);
   c2 = sum(x[,33:64] = 1);
   bad = abs(6 - c1) + abs(6 - c2);
   %mend;
```

```
%macro bad; * The hard way with scalars;
   c1 = (x1  = 1) + (x2  = 1) + (x3  = 1) + (x4  = 1) + (x5  = 1) + (x6  = 1)
      + (x7  = 1) + (x8  = 1) + (x9  = 1) + (x10 = 1) + (x11 = 1) + (x12 = 1)
      + (x13 = 1) + (x14 = 1) + (x15 = 1) + (x16 = 1) + (x17 = 1) + (x18 = 1)
      + (x19 = 1) + (x20 = 1) + (x21 = 1) + (x22 = 1) + (x23 = 1) + (x24 = 1)
      + (x25 = 1) + (x26 = 1) + (x27 = 1) + (x28 = 1) + (x29 = 1) + (x30 = 1)
      + (x31 = 1) + (x32 = 1);
   c2 = (x33 = 1) + (x34 = 1) + (x35 = 1) + (x36 = 1) + (x37 = 1) + (x38 = 1)
      + (x39 = 1) + (x40 = 1) + (x41 = 1) + (x42 = 1) + (x43 = 1) + (x44 = 1)
      + (x45 = 1) + (x46 = 1) + (x47 = 1) + (x48 = 1) + (x49 = 1) + (x50 = 1)
      + (x51 = 1) + (x52 = 1) + (x53 = 1) + (x54 = 1) + (x55 = 1) + (x56 = 1)
      + (x57 = 1) + (x58 = 1) + (x59 = 1) + (x60 = 1) + (x61 = 1) + (x62 = 1)
      + (x63 = 1) + (x64 = 1);
   bad = abs(6 - c1) + abs(6 - c2);
   %mend;
```

Both count how many times the first 32 attributes are different from 1 and how many times the second 32 attributes are different from 1. The first macro does so by comparing a vector with a scalar resulting in a vector of ones when the comparison is true and zeros when the comparison is false. Both create a sum of a series of zeros and ones (falses and trues). Both increase badness when there are not 6 ones in the first 32 attributes and 6 ones in the second 32 attributes.

Example 3, Searching a Candidate Set of Alternatives

In this example, we create a design for the same study as in the previous example. The product line is the same three fictitious breakfast bars with the same attributes as before. However, this time, rather than making a design in which all of the attributes of all of the alternatives are balanced and orthogonal, we instead make a design that is efficient for a choice model under the null hypothesis $\boldsymbol{\beta} = \mathbf{0}$ and with a specific model specification. The design is constructed from a candidate set of alternatives. See page 127 for more information about the brands and levels.

Our design consists of sets of alternatives, just like the design in the previous example (see page 134). In each set, the first alternative always consists of one of the 8 combinations for Branolicious (4 prices $\times$ 2 sizes). Similarly, the second alternative always consists of one of the 8 combinations for Brantopia, and the third alternative always consists of one of the 8 combinations for Brantasia. There is only one possibility for the constant or "None" alternative. We construct a candidate set that consists of four types of candidates, one type for each of the four alternative brands, and then we use the `%ChoicEff` macro to build a design from those candidates.

First, we use `%MktEx` to make the full set of price and size combinations that are needed to make the candidate design. The `%MktEx` specification makes a 4×2 experimental design in 8 runs. If this example had been substantially larger, we might instead make an orthogonal array with a subset of the combinations instead of a full-factorial design. We use the `%MktLab` macro to provide the names of the attributes. The original (not the randomized) design from the `%MktEx` macro is used as input. (Since all combinations are being used, there is no need to randomize.) The following steps make the design:

```
title 'Cereal Bars';

%mktex(4 2,                        /* all attribute levels                */
       n=8)                        /* number of candidate alternatives    */

%mktlab(data=design,               /* original design from MktEx          */
        vars=Price Count)          /* new variable names                  */

proc print; run;
```

The results are as follows:

```
                              Cereal Bars

                          Obs    Price    Count

                           1       1        1
                           2       1        2
                           3       2        1
                           4       2        2
                           5       3        1
                           6       3        2
                           7       4        1
                           8       4        2
```

From this design, we can create the full list of candidates as follows:

```
data cand;
   length Brand $ 12;
   retain Price Count . f1-f4 0;

   if _n_ = 1 then do;
      brand = 'None        '; f4 = 1; output; f4 = 0;    /* brand 4 (None)  */
      end;

   set final;

   brand = 'Branolicious';    f1 = 1; output; f1 = 0;    /* brand 1         */
   brand = 'Brantasia   ';    f2 = 1; output; f2 = 0;    /* brand 2         */
   brand = 'Brantopia   ';    f3 = 1; output; f3 = 0;    /* brand 3         */
   run;

proc print; run;
```

This DATA step reads each row of the full-factorial design and processes it. However, before it executes the `set` statement for the first time, on the first pass through the DATA step (when `_n_ = 1`), it writes out a candidate for the None alternative and flags it by changing `f4` to 1. Note that initially, `f1-f4` are all initialized to zero in the `retain` statement. Then `f4` is set back to zero after the row is written to the SAS data set. When each of the 8 price and size combinations is read in, three are written out, one for each brand. When a Branolicious alternative is created, `f1` is set to 1 and `f2`, `f3`, and `f4` are zero. When a Brantasia alternative is created, `f2` is set to 1 and `f1`, `f3`, and `f4` are zero. When a Brantopia alternative is created, `f3` is set to 1 and `f1`, `f2`, and `f4` are zero. The results are as follows:

```
                 Obs    Brand           Price    Count    f1    f2    f3    f4

                   1    None              .        .       0     0     0     1
                   2    Branolicious      1        1       1     0     0     0
                   3    Brantasia         1        1       0     1     0     0
                   4    Brantopia         1        1       0     0     1     0

                   5    Branolicious      1        2       1     0     0     0
                   6    Brantasia         1        2       0     1     0     0
                   7    Brantopia         1        2       0     0     1     0

                   8    Branolicious      2        1       1     0     0     0
                   9    Brantasia         2        1       0     1     0     0
                  10    Brantopia         2        1       0     0     1     0

                  11    Branolicious      2        2       1     0     0     0
                  12    Brantasia         2        2       0     1     0     0
                  13    Brantopia         2        2       0     0     1     0

                  14    Branolicious      3        1       1     0     0     0
                  15    Brantasia         3        1       0     1     0     0
                  16    Brantopia         3        1       0     0     1     0

                  17    Branolicious      3        2       1     0     0     0
                  18    Brantasia         3        2       0     1     0     0
                  19    Brantopia         3        2       0     0     1     0

                  20    Branolicious      4        1       1     0     0     0
                  21    Brantasia         4        1       0     1     0     0
                  22    Brantopia         4        1       0     0     1     0

                  23    Branolicious      4        2       1     0     0     0
                  24    Brantasia         4        2       0     1     0     0
                  25    Brantopia         4        2       0     0     1     0
```

The result is a data set with $3 \times 8 + 1 = 25$ candidates. The flag variables, `f1-f4`, designate the first alternative (`f1` = 1), second alternative (`f2` = 1), third alternative (`f3` = 1), and fourth alternative (`f4` = 1). In this study, exactly one variable in the `f1-f4` list is equal to 1 at any one time. If this were a generic study with no brands and the same attribute levels for each alternative, it would be possible to use the same candidate for multiple alternatives. The results might look clearer when sorted by brand. Sorting is not necessary. It just permits a clearer display of the structure of the data set. Formats are also provided. The following steps process and display the candidate set:

```
proc format;
   value price 1 = $2.89 2 = $2.99 3 = $3.09 4 = $3.19 . = ' ';
   value count 1 = 'Six Bars' 2 = 'Eight Bars'  . = ' ';
   run;
```

```
proc sort;
   by brand price count;
   format price price. count count.;
   run;

proc print label; run;
```

The results, sorted by brand, are as follows:

```
                              Cereal Bars

          Obs    Brand           Price      Count        f1    f2    f3    f4

            1    Branolicious    $2.89    Six Bars        1     0     0     0
            2    Branolicious    $2.89    Eight Bars      1     0     0     0
            3    Branolicious    $2.99    Six Bars        1     0     0     0
            4    Branolicious    $2.99    Eight Bars      1     0     0     0
            5    Branolicious    $3.09    Six Bars        1     0     0     0
            6    Branolicious    $3.09    Eight Bars      1     0     0     0
            7    Branolicious    $3.19    Six Bars        1     0     0     0
            8    Branolicious    $3.19    Eight Bars      1     0     0     0
            9    Brantasia       $2.89    Six Bars        0     1     0     0
           10    Brantasia       $2.89    Eight Bars      0     1     0     0
           11    Brantasia       $2.99    Six Bars        0     1     0     0
           12    Brantasia       $2.99    Eight Bars      0     1     0     0
           13    Brantasia       $3.09    Six Bars        0     1     0     0
           14    Brantasia       $3.09    Eight Bars      0     1     0     0
           15    Brantasia       $3.19    Six Bars        0     1     0     0
           16    Brantasia       $3.19    Eight Bars      0     1     0     0
           17    Brantopia       $2.89    Six Bars        0     0     1     0
           18    Brantopia       $2.89    Eight Bars      0     0     1     0
           19    Brantopia       $2.99    Six Bars        0     0     1     0
           20    Brantopia       $2.99    Eight Bars      0     0     1     0
           21    Brantopia       $3.09    Six Bars        0     0     1     0
           22    Brantopia       $3.09    Eight Bars      0     0     1     0
           23    Brantopia       $3.19    Six Bars        0     0     1     0
           24    Brantopia       $3.19    Eight Bars      0     0     1     0
           25    None                                     0     0     0     1
```

The candidate set consists of eight rows for the first alternative, flagged by f1 = 1 (and f2 = f3 = f4 = 0). The second group is flagged by f2 = 1, and so on. The constant alternative is flagged by f4 = 1. The design is created from the candidates using the %ChoicEff macro as follows:

```
%choiceff(data=cand,                /* candidate set of alternatives       */
          bestout=sasuser.cerealdes,/* choice design permanently stored    */
                                    /* model with stdz orthogonal coding   */
          model=class(brand price count / sta),
          maxiter=10,               /* maximum number of designs to make   */
          flags=f1-f4,              /* flag which alt can go where, 4 alts */
          nsets=16,                 /* number of choice sets               */
          seed=306,                 /* random number seed                  */
          options=relative,         /* display relative D-efficiency       */
          beta=zero)                /* assumed beta vector, Ho: b=0        */
```

The data= option names the data set of candidates. The best design is stored in a SAS data set named in the bestout= option. In the %ChoicEff macro, all of the designs go to the out= data set, so typically you want to use the bestout= data set instead. By default, this data set is called best. Here we create a permanent SAS data set so that it is around at analysis time. The main-effects model is specified in the model= option, and a standardized orthogonal coding is used. When you have a candidate set of alternatives, as we have here, then you need to specify the flag variables in the flags= option. Otherwise, when you have a candidate set of choice sets, you specify the nalts= option. Note that the %ChoicEff macro uses the number of variables in the flags= list to set the number of alternatives. The beta=zero option specifies the assumed parameter vector. We specify a random number seed so that we always get the same design if we rerun the %ChoicEff macro. Some of the results are as follows:

```
                              Cereal Bars

                             Final Results

                        Design                 4
                        Choice Sets           16
                        Alternatives           4
                        Parameters             7
                        Maximum Parameters    48
                        D-Efficiency     12.9142
                        Relative D-Eff   80.7140
                        D-Error           0.0774
                        1 / Choice Sets   0.0625
```

```
                                   Cereal Bars                                 16

                                                                           Standard
    n    Variable Name        Label                     Variance    DF       Error

    1    BrandBranolicious    Brand Branolicious        0.062500     1     0.25000
    2    BrandBrantasia       Brand Brantasia           0.062500     1     0.25000
    3    BrandBrantopia       Brand Brantopia           0.062500     1     0.25000
    4    Price_2_89           Price $2.89               0.090933     1     0.30155
    5    Price_2_99           Price $2.99               0.090980     1     0.30163
    6    Price_3_09           Price $3.09               0.090909     1     0.30151
    7    CountSix_Bars        Count Six Bars            0.091003     1     0.30167
                                                                    ==
                                                                     7
```

This table shows the variances and standard errors under the null-hypothesis assumption $\boldsymbol{\beta} = \mathbf{0}$. We see three parameters for brand (4 alternatives including None minus 1), three for price $(4 - 1)$, one for count $(2 - 1)$. With 16 choice sets and 4 alternatives, we can estimate at most $16 \times (4 - 1) = 48$ parameters. All are estimable, and all have reasonable standard errors. The variances for the brand effects are the inverse of the number of choice sets $(1/16 = 0.06250)$.[*] The other variances are bigger. These results look good.

The following step displays the choice sets:

```
   proc print data=sasuser.cerealdes;
      by set;
      id set;
      var brand -- count;
      run;
```

The choice sets are as follows:

```
                                   Cereal Bars

                        Set    Brand           Price      Count

                          1    Branolicious    $3.19    Six Bars
                               Brantasia       $2.89    Six Bars
                               Brantopia       $2.99    Eight Bars
                               None

                          2    Branolicious    $2.89    Eight Bars
                               Brantasia       $3.19    Six Bars
                               Brantopia       $2.99    Eight Bars
                               None
```

[*]This comparison is only valid when the standardized orthogonal contrast coding is used.

```
  3     Branolicious    $3.19    Six Bars
        Brantasia       $3.09    Eight Bars
        Brantopia       $2.99    Six Bars
        None

  4     Branolicious    $2.99    Eight Bars
        Brantasia       $3.19    Six Bars
        Brantopia       $2.89    Six Bars
        None

  5     Branolicious    $3.09    Six Bars
        Brantasia       $2.99    Eight Bars
        Brantopia       $2.89    Eight Bars
        None

  6     Branolicious    $3.09    Eight Bars
        Brantasia       $3.19    Eight Bars
        Brantopia       $2.89    Six Bars
        None

  7     Branolicious    $2.99    Eight Bars
        Brantasia       $2.89    Six Bars
        Brantopia       $3.09    Six Bars
        None

  8     Branolicious    $2.89    Eight Bars
        Brantasia       $3.09    Six Bars
        Brantopia       $2.99    Six Bars
        None

  9     Branolicious    $3.09    Six Bars
        Brantasia       $2.89    Eight Bars
        Brantopia       $3.19    Eight Bars
        None

 10     Branolicious    $2.89    Eight Bars
        Brantasia       $2.99    Six Bars
        Brantopia       $3.19    Eight Bars
        None

 11     Branolicious    $3.19    Eight Bars
        Brantasia       $3.09    Eight Bars
        Brantopia       $2.89    Six Bars
        None

 12     Branolicious    $3.09    Eight Bars
        Brantasia       $2.99    Six Bars
        Brantopia       $3.19    Eight Bars
        None
```

```
13    Branolicious    $2.89    Six Bars
      Brantasia       $3.19    Six Bars
      Brantopia       $3.09    Eight Bars
      None

14    Branolicious    $2.99    Six Bars
      Brantasia       $2.89    Eight Bars
      Brantopia       $3.09    Six Bars
      None

15    Branolicious    $3.19    Six Bars
      Brantasia       $2.99    Eight Bars
      Brantopia       $3.09    Six Bars
      None

16    Branolicious    $2.99    Six Bars
      Brantasia       $3.09    Eight Bars
      Brantopia       $3.19    Eight Bars
      None
```

It is instructive to look at the covariance matrix of the parameter estimates. The following steps display this matrix, which is automatically output to a SAS data set called `bestcov`:

```
proc format;
   value zer -1e-12 - 1e-12 = ' 0   ';
   run;

proc print data=bestcov label;
   id __label;
   label __label = '00'x;
   var BrandBranolicious -- CountSix_Bars;
   format _numeric_ zer5.2;
   run;
```

The format displays values very close to zero as precisely zero to make a better display. The results are as follows:

```
                                   Cereal Bars

                          Brand       Brand     Brand   Price Price Price  Count
                       Branolicious Brantasia Brantopia $2.89 $2.99 $3.09 Six Bars

    Brand Branolicious     0.06        0         0       0     0     0      0
    Brand Brantasia        0           0.06      0       0     0     0      0
    Brand Brantopia        0           0         0.06    0     0     0      0
    Price $2.89            0           0         0       0.09  0.00  0      0.00
    Price $2.99            0           0         0       0.00  0.09  0      0.00
    Price $3.09            0           0         0       0     0     0.09   0
    Count Six Bars         0           0         0       0.00  0.00  0      0.09
```

There are some nonzero covariances between the price and count attributes.

There is one more test that should be run before a design is used. In the following step, the `%MktDups` macro checks the design to see if any choice sets are duplicates of any other choice sets:

```
%mktdups(branded,                         /* a design with brands           */
         data=sasuser.cerealdes,          /* the input design to evaluate   */
         factors=brand price count,       /* factors in the design          */
         nalts=4)                         /* number of alternatives         */
```

The results are as follows:

```
Design:          Branded
Factors:         brand price count
                 Brand
                 Count Price
Duplicate Sets:  0
```

The first line of the table tells us that this is a branded design as opposed to a generic design (bundles of attributes with no brands). The second line tells us the factors as specified in the `factors=` option. These are followed by the actual variable names for the factors. The last line reports the number of duplicates. In this case, there are no duplicate choice sets. If there had been duplicate choice sets, then changing the random number seed might help. Sometimes, changing other aspects of the design or the approach for making the design helps.

The questionnaire is designed and data are collected and entered, as they were in the previous example. The data are entered in the following step:

```
title2 'Read Data';

data results;
   input Subject (r1-r16) (1.);
   datalines;
  1 3131312123331232
  2 3131332121331222
  3 3133331121321222
  4 3111232121321223
  5 3131312122311222
  6 3131333121213232
  7 3131311132331332
  8 3131331121321223
  9 3121332122331222
 10 3131132123321222
 11 3131312121321223
 12 3121332122311232
 13 3131331122321222
 14 3131311121331223
 15 3131311121121223
 16 3121333221321223
 17 3121332131311222
 18 3131233121331222
 19 3131332121131232
 20 3131231121311432
 21 2133232121331223
 22 3121332121121221
 23 3131332121331223
 24 3133312131311323
 25 3131332431311232
 26 3131312143331222
 27 3121112133311223
 28 3123331121321221
 29 3121312122311231
 30 3131332121311233
 31 3121332121341222
 32 2131231122331231
 33 2131312122231221
 34 2121332321311421
 35 3133311121331233
 36 3131333121321323
 37 3131312131331223
 38 3121312331331221
 39 3121312121221231
 40 3123232121321232
;
```

The %MktMerge macro merges the data and the design and creates the dependent variable as follows:

```
title2 'Merge Data and Design';

%mktmerge(design=sasuser.cerealdes, /* input design                        */
          data=results,             /* input data set                      */
          out=res2,                 /* output data set with design and data */
          nsets=16,                 /* number of choice sets               */
          nalts=4,                  /* number of alternatives              */
          setvars=r1-r16)           /* variables with the chosen alt nums  */
```

The coding and analysis are the same as in the previous example and are as follows:

```
title2 'Code the Independent Variables';

proc transreg design norestoremissing data=res2;
   model class(brand price count);
   id subject set c;
   output out=coded(drop=_type_ _name_ intercept) lprefix=0;
   run;

%phchoice(on)                       /* customize PHREG for a choice model  */

title2 'Multinomial Logit Discrete Choice Model';

proc phreg data=coded brief;
   model c*c(2) = &_trgind / ties=breslow;
   strata subject set;
   run;

%phchoice(off)                      /* restore PHREG to a survival PROC    */
```

The parameter estimate table is as follows:

```
                              Cereal Bars
                 Multinomial Logit Discrete Choice Model

                          The PHREG Procedure

                  Multinomial Logit Parameter Estimates

                          Parameter     Standard
                   DF      Estimate        Error    Chi-Square    Pr > ChiSq

      Branolicious  1       2.70313      0.48143       31.5257        <.0001
      Brantasia     1       2.32653      0.48515       22.9970        <.0001
      Brantopia     1       2.88085      0.47575       36.6674        <.0001
      $2.89         1       2.92298      0.20046      212.6164        <.0001
      $2.99         1       1.71761      0.18827       83.2303        <.0001
      $3.09         1       0.66151      0.19513       11.4922        0.0007
      Six Bars      1      -1.21635      0.11703      108.0246        <.0001
```

Example 4, Searching a Candidate Set of Alternatives with Restrictions

In this example, we create a design for the same study as the previous example. The product line is the same three fictitious breakfast bars with the same attributes as before. Like last time, we create a candidate set of alternatives and make a design that is efficient for a choice model under the null hypothesis $\boldsymbol{\beta} = \mathbf{0}$ and with a specific model specification. However, this time, we place restrictions on how the candidates can come into the design. This example uses options in the `%ChoicEff` macro that first became available with this edition of the macros and book.

We will begin by creating a candidate set of alternatives in exactly the same way that we did it in the preceding example:

```
title 'Cereal Bars';

%mktex(4 2,                        /* all attribute levels              */
       n=8)                        /* number of candidate alternatives  */

%mktlab(data=design,               /* original design from MktEx        */
        vars=Price Count)          /* new variable names                */

data cand;
   length Brand $ 12;
   retain Price Count . f1-f4 0;

   if _n_ = 1 then do;
      brand = 'None        '; f4 = 1; output; f4 = 0;    /* brand 4 (None)  */
      end;

   set final;

   brand = 'Branolicious';    f1 = 1; output; f1 = 0;    /* brand 1         */
   brand = 'Brantasia   ';    f2 = 1; output; f2 = 0;    /* brand 2         */
   brand = 'Brantopia   ';    f3 = 1; output; f3 = 0;    /* brand 3         */
   run;

proc format;
   value price 1 = $2.89 2 = $2.99 3 = $3.09 4 = $3.19 . = ' ';
   value count 1 = 'Six Bars' 2 = 'Eight Bars'  . = ' ';
   run;

proc sort;
   by brand price count;
   format price price. count count.;
   run;

proc print; run;
```

Sorting by brand is not necessary. It is just done here to more clearly show the structure of the candidate set. The candidate set consists of eight rows for the first alternative, flagged by `f1` = 1 (and `f2` = `f3` = `f4` = 0). The second group is flagged by `f2` = 1 and the rest zero, and so on. The constant alternative is flagged by `f4` = 1. The candidate set is as follows:

```
                              Cereal Bars

   Obs    Brand           Price      Count        f1    f2    f3    f4

     1    Branolicious    $2.89    Six Bars        1     0     0     0
     2    Branolicious    $2.89    Eight Bars      1     0     0     0
     3    Branolicious    $2.99    Six Bars        1     0     0     0
     4    Branolicious    $2.99    Eight Bars      1     0     0     0
     5    Branolicious    $3.09    Six Bars        1     0     0     0
     6    Branolicious    $3.09    Eight Bars      1     0     0     0
     7    Branolicious    $3.19    Six Bars        1     0     0     0
     8    Branolicious    $3.19    Eight Bars      1     0     0     0
     9    Brantasia       $2.89    Six Bars        0     1     0     0
    10    Brantasia       $2.89    Eight Bars      0     1     0     0
    11    Brantasia       $2.99    Six Bars        0     1     0     0
    12    Brantasia       $2.99    Eight Bars      0     1     0     0
    13    Brantasia       $3.09    Six Bars        0     1     0     0
    14    Brantasia       $3.09    Eight Bars      0     1     0     0
    15    Brantasia       $3.19    Six Bars        0     1     0     0
    16    Brantasia       $3.19    Eight Bars      0     1     0     0
    17    Brantopia       $2.89    Six Bars        0     0     1     0
    18    Brantopia       $2.89    Eight Bars      0     0     1     0
    19    Brantopia       $2.99    Six Bars        0     0     1     0
    20    Brantopia       $2.99    Eight Bars      0     0     1     0
    21    Brantopia       $3.09    Six Bars        0     0     1     0
    22    Brantopia       $3.09    Eight Bars      0     0     1     0
    23    Brantopia       $3.19    Six Bars        0     0     1     0
    24    Brantopia       $3.19    Eight Bars      0     0     1     0
    25    None                                     0     0     0     1
```

We will restrict the design to avoid choice sets where attributes are constant. That is, you want to select just the choice sets where neither price nor count is constant within a choice set (outside the constant alternative). This is accomplished with the following steps:

```
%macro res;
   if x[1,1] = x[2,1] & x[1,1] = x[3,1] then bad = 1;
   if x[1,2] = x[2,2] & x[1,2] = x[3,2] then bad = bad + 1;
   %mend;

%choiceff(data=cand,                /* candidate set of alternatives       */
          bestout=sasuser.cerealdes,/* choice design permanently stored    */
                                    /* model with stdz orthogonal coding   */
          model=class(brand price count / sta),
          maxiter=10,               /* maximum number of designs to make   */
          flags=f1-f4,              /* flag which alt can go where, 4 alts */
          nsets=16,                 /* number of choice sets               */
          seed=306,                 /* random number seed                  */
          options=relative          /* display relative D-efficiency       */
                  resrep,           /* detailed report on restrictions     */
          restrictions=res,         /* name of the restrictions macro      */
          resvars=price count,      /* vars used in defining restrictions  */
          beta=zero)                /* assumed beta vector, Ho: b=0        */
```

Two new options are used in the `%ChoiceEff` macro. The `resvars=` option lists the variables that are used to form restrictions. These variables must be numeric. Values of these variables are stored in a matrix called `x` for each choice set as it is being processed, and you can use these values to ensure that each choice set meets within choice set restrictions. Furthermore, while it is not illustrated in this example, the levels of the attributes for the entire design are stored in a matrix called `xmat`, and you can use these values as well as the values in `x` to impose restrictions across choice sets.

Restrictions are written with PROC IML statements embedded in a macro. You specify the name of the macro by using the `restrictions=` option. In the restrictions macro, you compute an IML scalar called `bad` that quantifies the badness of the design. In this case, since the restrictions are entirely within choice set, you can just quantify the badness of one choice set at at time by evaluating the values in `x`. The matrix `x` has four rows (since there are four alternatives) and two columns (since there are two variables listed in the `resvars=` option. The scalar `bad` is automatically initialized to zero by the `%ChoiceEff` macro. In this example, `bad` is set to 1 if the `Price` variable is constant in a choice set, `bad` is incremented by 1 if the `Count` variable is constant in a choice set. The `if` statements use the Boolean syntax of PROC IML. We must use the following IML logical operators, which do not have all of the same syntactical alternatives as DATA step operators:

Specify	For	Do Not Specify
=	equals	EQ
∧ = or ¬ =	not equals	NE
<	less than	LT
<=	less than or equal to	LE
>	greater than	GT
>=	greater than or equal to	GE
&	and	AND
|	or	OR
∧ or ¬	not	NOT
`a` <= `b` & `b` <= `c`	range check	`a` <= `b` <= `c`

One other option is specified in this example, and that is `options=resrep`. It is a good idea to specify this option with restrictions until you are sure that your restrictions macro is correct. It provides detailed information about the swaps that the `%ChoicEff` macro performs and its success in imposing restrictions.

Part of the iteration history table, with a lot of `options=resrep` information and other information deleted, is as follows:

```
                               Cereal Bars

               Design   Iteration  D-Efficiency      D-Error
               ----------------------------------------------
                    1       0          11.33805 *     0.08820
     at    1   1 swapped in     1      11.34298 bad =       0
     at    1   2 swapped in     4      11.34298 bad =       0
     at    1   2 swapped in     6      11.35439 bad =       0
     at    1   3 swapped in     7      11.46812 bad =       0
     at    1   4 swapped in     1      11.46812 bad =       0
     .
     .
     .
     at   16   4 swapped in     1      12.87489 bad =       0

     at    1   3 swapped in     8      12.87590 bad =       0
     .
     .
     .
     at    1   4 swapped in     1      12.87590 bad =       0
     at   15   4 swapped in     1      12.90851 bad =       0
                            2          12.90851 *     0.07747

               Design   Iteration  D-Efficiency      D-Error
               ----------------------------------------------
                    2       0          11.26776       0.08875
     at    1   1 swapped in     1      11.46996 bad =       0
     .
     .
     .
     at    1   2 swapped in     5      11.46996 bad =       0
     at   16   4 swapped in     1      12.87774 bad =       0
                            1          12.87774       0.07765
     at    1   1 swapped in     1      12.87774 bad =       0
     .
     .
     .
     at   10   4 swapped in     1      12.89796 bad =       0
                            2          12.89796       0.07753
     .
     .
     .
```

```
                 Design   Iteration  D-Efficiency       D-Error
                 ----------------------------------------------
                     10       0          10.41643       0.09600
       at    1   1 swapped in     1      10.64012 bad =       0
       .
       .
       .
       at   16   4 swapped in     1      12.85431 bad =       0
                                  1      12.85431       0.07779
       at    1   1 swapped in     1      12.85431 bad =       0
       .
       .
       .
       at    1   2 swapped in     4      12.85431 bad =       0
       at   16   4 swapped in     1      12.91615 bad =       0
                                  2      12.91615 *     0.07742
```

The first line shows the *D*-efficiency of the first initial design. The next row, and all other rows that begin with “`at`”, are produced by `options=resrep` and report the set and alternative that is being swapped, the number of the candidate alternative that is being swapped in (in this case it is candidate number within candidate type), the *D*-efficiency, and the badness after the swap. In this problem, you can see that the `%ChoicEff` macro has no problem minimizing the badness to zero. That is not always the case depending on both the design requirements and how you pose the restrictions. Among rows that do not begin with “`at`”, an asterisk is used to indicate places where *D*-efficiency is greater than any previously reported value.

The final results are as follows:

```
                                 Cereal Bars

                                Final Results

                          Design                10
                          Choice Sets           16
                          Alternatives           4
                          Parameters             7
                          Maximum Parameters    48
                          D-Efficiency     12.9161
                          Relative D-Eff   80.7259
                          D-Error           0.0774
                          1 / Choice Sets   0.0625
```

```
 n    Variable Name        Label                   Variance    DF      Error

 1    BrandBranolicious    Brand Branolicious      0.062500     1     0.25000
 2    BrandBrantasia       Brand Brantasia         0.062500     1     0.25000
 3    BrandBrantopia       Brand Brantopia         0.062500     1     0.25000
 4    Price_2_89           Price $2.89             0.090909     1     0.30151
 5    Price_2_99           Price $2.99             0.090909     1     0.30151
 6    Price_3_09           Price $3.09             0.090909     1     0.30151
 7    CountSix_Bars        Count Six Bars          0.090909     1     0.30151
                                                               ==
                                                                7
```

It is instructive to look at the covariance matrix of the parameter estimates. The following steps display this matrix, which is automatically output to a SAS data set called `bestcov`:

```
proc format;
   value zer -1e-12 - 1e-12 = ' 0   ';
   run;

proc print data=bestcov label;
   id __label;
   label __label = '00'x;
   var BrandBranolicious -- CountSix_Bars;
   format _numeric_ zer5.2;
   run;
```

The results are as follows:

```
                                     Cereal Bars

                         Brand        Brand     Brand   Price Price Price  Count
                      Branolicious Brantasia Brantopia $2.89 $2.99 $3.09 Six Bars

    Brand Branolicious     0.06         0         0       0     0     0       0
    Brand Brantasia        0            0.06      0       0     0     0       0
    Brand Brantopia        0            0         0.06    0     0     0       0
    Price $2.89            0            0         0       0.09  0     0       0
    Price $2.99            0            0         0       0     0.09  0       0
    Price $3.09            0            0         0       0     0     0.09    0
    Count Six Bars         0            0         0       0     0     0       0.09
```

With a diagonal covariance matrix, this design appears optimal for this problem. If this design is optimal, you might wonder why relative *D*-efficiency is not 100%. The variances for both price and count are larger than the variances for brand. A relative *D*-efficiency of 100% is based on a hypothetical optimal design where every factor can achieve the minimum variance like the brand attribute. Here, we have two- and three-level factors in a choice set with four alternatives, so perfect balance and hence perfect *D*-efficiency is not possible. Furthermore, we have a constant alternative, which places another constraint on the maximum *D*-efficiency.

The following step prints the design:

```
   proc print data=sasuser.cerealdes; id set; by set; var brand price count; run;
```

The design is as follows:

```
                              Cereal Bars

                 Set    Brand           Price      Count

                   1    Branolicious    $2.89    Six Bars
                        Brantasia       $2.99    Eight Bars
                        Brantopia       $3.09    Six Bars
                        None

                   2    Branolicious    $2.89    Six Bars
                        Brantasia       $3.19    Six Bars
                        Brantopia       $2.99    Eight Bars


                   3    Branolicious    $3.19    Eight Bars
                        Brantasia       $2.99    Eight Bars
                        Brantopia       $2.89    Six Bars
                        None

                   4    Branolicious    $3.19    Six Bars
                        Brantasia       $3.09    Six Bars
                        Brantopia       $2.99    Eight Bars
                        None

                   5    Branolicious    $3.09    Eight Bars
                        Brantasia       $2.89    Eight Bars
                        Brantopia       $2.99    Six Bars
                        None

                   6    Branolicious    $3.09    Six Bars
                        Brantasia       $2.89    Eight Bars
                        Brantopia       $3.19    Eight Bars
                        None

                   7    Branolicious    $3.09    Eight Bars
                        Brantasia       $3.19    Six Bars
                        Brantopia       $2.99    Six Bars
                        None

                   8    Branolicious    $2.99    Eight Bars
                        Brantasia       $3.09    Six Bars
                        Brantopia       $2.89    Eight Bars
                        None
```

```
                              9    Branolicious    $3.19    Eight Bars
                                   Brantasia       $3.09    Six Bars
                                   Brantopia       $2.89    Six Bars
                                   None

                             10    Branolicious    $2.89    Six Bars
                                   Brantasia       $3.19    Eight Bars
                                   Brantopia       $3.09    Eight Bars
                                   None

                             11    Branolicious    $2.99    Six Bars
                                   Brantasia       $3.09    Eight Bars
                                   Brantopia       $3.19    Eight Bars
                                   None

                             12    Branolicious    $3.19    Six Bars
                                   Brantasia       $2.89    Eight Bars
                                   Brantopia       $3.09    Six Bars
                                   None

                             13    Branolicious    $2.99    Six Bars
                                   Brantasia       $2.89    Eight Bars
                                   Brantopia       $3.19    Eight Bars
                                   None

                             14    Branolicious    $3.09    Eight Bars
                                   Brantasia       $2.99    Six Bars
                                   Brantopia       $2.89    Six Bars
                                   None

                             15    Branolicious    $2.89    Eight Bars
                                   Brantasia       $2.99    Six Bars
                                   Brantopia       $3.19    Six Bars
                                   None

                             16    Branolicious    $2.99    Eight Bars
                                   Brantasia       $3.19    Six Bars
                                   Brantopia       $3.09    Eight Bars
                                   None
```

You can see that there are no constant attributes.

There is one more test that should be run before a design is used. In the following step, the `%MktDups` macro checks the design to see if any choice sets are duplicates of any other choice sets:

```
%mktdups(branded,                     /* a design with brands          */
         data=sasuser.cerealdes,      /* the input design to evaluate  */
         factors=brand price count,   /* factors in the design         */
         nalts=4)                     /* number of alternatives        */
```

The results are as follows:

```
Design:          Branded
Factors:         brand price count
                 Brand
                 Count Price
Duplicate Sets:  0
```

You can see that there are no duplicate choice sets.

The questionnaire is designed and data are collected and entered, as they were in the previous example. The data are entered in the following step:

```
title2 'Read Data';

data results;
   input Subject (r1-r16) (1.);
   datalines;
  1 2133321333322311
  2 3133221131222211
  3 2133321131122311
  4 1113221131322311
  5 2133321133322311
  6 2133221131222311
  7 2133221132322311
  8 2133221131322311
  9 1123221112222311
 10 1133123133222211
 11 3133221331322111
 12 1123221113223311
 13 2231221132123321
 14 2133221331222321
 15 2133321131122111
 16 2123221331222321
 17 2123223331332311
 18 2133223311132121
 19 1133221131122311
 20 2133221131222411
 21 2133221131222321
 22 2123221111122321
 23 2133223131222111
 24 1133321111332111
 25 3133321431222131
 26 2133221143223113
 27 2123131113322311
 28 3322221131322311
 29 2123221133322311
 30 2133221131222311
 31 1121221111242311
 32 2133221133122311
 33 2133321133222311
 34 2123221331322411
 35 1132221111322311
 36 1133323131322121
 37 1333221131322311
 38 1123321311222311
 39 1123223131222111
 40 2123221331222313
;
```

The `%MktMerge` macro merges the data and the design and creates the dependent variable as follows:

```
title2 'Merge Data and Design';

%mktmerge(design=sasuser.cerealdes, /* input design                        */
          data=results,             /* input data set                      */
          out=res2,                 /* output data set with design and data */
          nsets=16,                 /* number of choice sets               */
          nalts=4,                  /* number of alternatives              */
          setvars=r1-r16)           /* variables with the chosen alt nums  */
```

The coding and analysis are the same as in the previous example and are as follows:

```
title2 'Code the Independent Variables';

proc transreg design norestoremissing data=res2;
   model class(brand price count);
   id subject set c;
   output out=coded(drop=_type_ _name_ intercept) lprefix=0;
   run;

%phchoice(on)                       /* customize PHREG for a choice model  */

title2 'Multinomial Logit Discrete Choice Model';

proc phreg data=coded brief;
   model c*c(2) = &_trgind / ties=breslow;
   strata subject set;
   run;

%phchoice(off)                      /* restore PHREG to a survival PROC    */
```

The parameter estimate table is as follows:

```
                        Multinomial Logit Parameter Estimates

                                 Parameter      Standard
                        DF        Estimate         Error    Chi-Square    Pr > ChiSq

      Branolicious       1         2.88724       0.48197       35.8866        <.0001
      Brantasia          1         2.43226       0.49040       24.5995        <.0001
      Brantopia          1         2.41739       0.48561       24.7815        <.0001
      $2.89              1         2.91685       0.19962      213.5135        <.0001
      $2.99              1         1.73771       0.19894       76.2957        <.0001
      $3.09              1         0.69198       0.21241       10.6125        0.0011
      Six Bars           1        -1.28398       0.12586      104.0674        <.0001
```

These results are similar to what we saw in the previous example.

Example 5, Searching a Candidate Set of Choice Sets

In this example, we create a design for the same study as the previous example. The product line is the same three fictitious breakfast bars with the same attributes as before. However, this time, rather than making a design from candidate alternatives, we create a design from candidate choice sets. Usually, you would not use this approach, but you could use it when there are restrictions that prevent certain alternatives from appearing with other alternatives. You provide candidate choice sets to the `%ChoicEff` macro that conform to the restrictions. The candidate alternative approach in the previous example will usually be superior to the candidate choice set approach in this example since the former can consider more possible designs using smaller candidate sets. However, for very small problems such as this one, the two approaches should perform similarly.

We will begin like we began the first example, by making a design that is *D*-efficient for a linear model and then converting it to a choice design. However, this time, that design will form a candidate set and not the final design. We can use the `%MktRuns` macro as follows to suggest the number of choice sets in the candidate set. The input to the macro is the number of levels of all of the factors (that is, all of the attributes of all of the alternatives). The following step runs the macro:

```
   title 'Cereal Bars';

   %mktruns(4 2  4 2  4 2)     /* factor level list for all attrs and alts     */
```

The results are as follows:

```
                          Cereal Bars

                        Design Summary

                    Number of
                    Levels       Frequency

                         2           3
                         4           3

                          Cereal Bars

             Saturated      = 13
             Full Factorial = 512

             Some Reasonable                     Cannot Be
                Design Sizes       Violations    Divided By

                          16 *              0
                          32 *              0
                          24                3     16
                          20               12      8 16
                          28               12      8 16
                          14               18      4  8 16
                          18               18      4  8 16
                          22               18      4  8 16
                          26               18      4  8 16
                          30               18      4  8 16
                          13 S             21      2  4  8 16

        * - 100% Efficient design can be made with the MktEx macro.
        S - Saturated Design - The smallest design that can be made.
            Note that the saturated design is not one of the
            recommended designs for this problem.  It is shown
            to provide some context for the recommended sizes.
```

```
                              Cereal Bars

     n                   Design                         Reference

    16    2 **  6         4 **  3               Fractional-Factorial
    16    2 **  3         4 **  4               Fractional-Factorial
    32    2 ** 22         4 **  3               Fractional-Factorial
    32    2 ** 19         4 **  4               Fractional-Factorial
    32    2 ** 16         4 **  5               Fractional-Factorial
    32    2 ** 15         4 **  3   8 **  1     Fractional-Factorial
    32    2 ** 13         4 **  6               Fractional-Factorial
    32    2 ** 12         4 **  4   8 **  1     Fractional-Factorial
    32    2 ** 10         4 **  7               Fractional-Factorial
    32    2 **  9         4 **  5   8 **  1     Fractional-Factorial
    32    2 **  7         4 **  8               Fractional-Factorial
    32    2 **  6         4 **  6   8 **  1     Fractional-Factorial
    32    2 **  4         4 **  9               Fractional-Factorial
    32    2 **  3         4 **  7   8 **  1     Fractional-Factorial
```

Since the goal is to make a candidate set of choice sets not the final design, we can pick a much larger number of choice sets than we used in the first example. The `%MktRuns` macro lists orthogonal arrays that the `%MktEx` macro knows how to make. In some cases, these might provide good candidate designs. However, in this case, the full-factorial design, at 512 runs, is not too large for this problem. We use the `%MktEx` macro as follows to create the full-factorial design:

```
%mktex(4 2  4 2  4 2,      /* factor level list for all attrs and alts     */
       n=512,              /* number of candidate choice sets              */
       out=full,           /* output data set with full factorial          */
       seed=17)            /* random number seed                           */
```

The results are as follows:

```
                              Cereal Bars

                        Algorithm Search History

                             Current          Best
          Design    Row,Col  D-Efficiency  D-Efficiency  Notes
          ----------------------------------------------------------
               1      Start      100.0000      100.0000  Tab
               1        End      100.0000
```

```
                              Cereal Bars

                          The OPTEX Procedure

                        Class Level Information

                        Class  Levels  Values

                        x1       4     1 2 3 4
                        x2       2     1 2
                        x3       4     1 2 3 4
                        x4       2     1 2
                        x5       4     1 2 3 4
                        x6       2     1 2

                              Cereal Bars
                                                                   Average
                                                                  Prediction
       Design                                                      Standard
       Number    D-Efficiency     A-Efficiency     G-Efficiency       Error
       ------------------------------------------------------------------------
            1       100.0000         100.0000         100.0000          0.1593
```

The full-factorial design has 100% *D*-efficiency.

Say that you want to restrict the design to avoid choice sets where attributes are constant. That is, you want to select just the choice sets where neither price nor count is constant within a choice set. The following step subsets the design by deleting choice sets with constant alternatives:

```
data design;
   set full;
   if x1 eq x3 and x3 eq x5 then delete;  /* delete constant price          */
   if x2 eq x4 and x4 eq x6 then delete;  /* delete constant count          */
   run;
```

Note that it is easier to impose restrictions across alternatives at this stage (when the design is arranged with one row per choice set) rather than later (when the design is arranged with one row per alternative per choice set). Now, restrictions can be imposed by examining the variables in one row at a time rather than looking across multiple rows. After running this step, the candidate design has 360 choice sets. You can also specify restrictions directly in the `%MktEx` macro. This is illustrated extensively throughout this book, but it is beyond the scope of this chapter. For this problem, the DATA step approach is superior, since the goal is to eliminate unsuitable candidates from a full-factorial design. Usually, the restrictions are not this simple.

Like the linear arrangement example, we can use the `%MktKey` macro and the `%MktRoll` macro to create the key to converting the linear arrangement into a choice design.

The following steps create and display the key data set:

```
%mktkey(3 2)         /* x1-x6 (since 3*2=6) in 3 rows and 2 columns         */

data key;
   input Brand $ 1-12 Price $ Count $;
   datalines;
Branolicious    x1         x2
Brantopia       x3         x4
Brantasia       x5         x6
None            .          .
;

proc print; run;
```

The results are as follows:

```
                                   Cereal Bars

                        Obs    Brand           Price    Count

                         1     Branolicious     x1       x2
                         2     Brantopia        x3       x4
                         3     Brantasia        x5       x6
                         4     None
```

The following steps create the candidate set of choice sets from the linear arrangement using the rules specified in the `key=key` data set, create and set the formats, and display the first four candidate choice sets:

```
%mktroll(design=design,      /* input linear candidate design               */
         key=key,            /* rules for making choice design              */
         alt=brand,          /* brand or alternative label var              */
         out=cand)           /* output candidate choice design              */

proc format;
   value price 1 = $2.89 2 = $2.99 3 = $3.09 4 = $3.19 . = ' ';
   value count 1 = 'Six Bars' 2 = 'Eight Bars'  . = ' ';
   run;

data cand;
   set cand;
   format price price. count count.;
   run;

proc print data=cand(obs=16); id set; by set; run;
```

The first four choice sets are as follows:

```
                              Cereal Bars

              Set    Brand           Price      Count

               1     Branolicious    $2.89    Six Bars
                     Brantopia       $2.89    Six Bars
                     Brantasia       $2.99    Eight Bars
                     None

               2     Branolicious    $2.89    Six Bars
                     Brantopia       $2.89    Six Bars
                     Brantasia       $3.09    Eight Bars
                     None

               3     Branolicious    $2.89    Six Bars
                     Brantopia       $2.89    Six Bars
                     Brantasia       $3.19    Eight Bars
                     None

               4     Branolicious    $2.89    Six Bars
                     Brantopia       $2.89    Eight Bars
                     Brantasia       $2.99    Six Bars
                     None
```

You can see that in these choice sets that all of the attributes vary. The full candidate data set contains 1440 observations, from the 360 candidates each with 4 alternatives. Next, like the example with a candidate set of alternatives, the `%ChoicEff` macro is run to create a design that is efficient under the null hypothesis $\boldsymbol{\beta} = \mathbf{0}$ and with a main-effects model specification. However, this time there are no flag variables or `flags=` option to indicate which alternatives go where, since the candidate set contains choice sets not alternatives. Instead, the `nalts=` option is used to specify the number of alternatives in each choice set. The `%ChoicEff` macro assumes that each block of `nalts=4` alternatives forms one candidate choice set. Again, a standardized orthogonal contrast coding is used. By default, the best design found is stored in the `outbest=best` SAS data set. The following step creates the design:

```
%choiceff(data=cand,                      /* candidate set of choice sets   */
                                          /* model with stdz orthog coding  */
          model=class(brand price count / sta) /
                cprefix=0                 /* lpr=0 labels from just levels  */
                lprefix=0,                /* cpr=0 names from just levels   */
          nsets=16,                       /* number of choice sets          */
          seed=145,                       /* random number seed             */
          nalts=4,                        /* number of alternatives         */
          options=relative,               /* display relative D-efficiency  */
          beta=zero)                      /* assumed beta vector            */
```

The results are as follows:

```
                             Cereal Bars

          n    Name            Beta    Label

          1    Branolicious      0     Branolicious
          2    Brantasia         0     Brantasia
          3    Brantopia         0     Brantopia
          4    _2_89             0     $2.89
          5    _2_99             0     $2.99
          6    _3_09             0     $3.09
          7    Six_Bars          0     Six Bars

        Design   Iteration  D-Efficiency       D-Error
        ----------------------------------------------
             1       0          11.19099 *     0.08936
                     1          12.91424 *     0.07743
                     2          12.91424       0.07743

        Design   Iteration  D-Efficiency       D-Error
        ----------------------------------------------
             2       0          11.50095       0.08695
                     1          12.88929       0.07758
                     2          12.91042       0.07746

                             Cereal Bars

                            Final Results

                  Design                   1
                  Choice Sets             16
                  Alternatives             4
                  Parameters               7
                  Maximum Parameters      48
                  D-Efficiency       12.9142
                  Relative D-Eff     80.7140
                  D-Error             0.0774
                  1 / Choice Sets     0.0625
```

```
                                  Cereal Bars

            Variable                                                 Standard
       n    Name              Label             Variance     DF        Error

       1    Branolicious      Branolicious      0.062500      1      0.25000
       2    Brantasia         Brantasia         0.062500      1      0.25000
       3    Brantopia         Brantopia         0.062500      1      0.25000
       4    _2_89             $2.89             0.090933      1      0.30155
       5    _2_99             $2.99             0.090917      1      0.30152
       6    _3_09             $3.09             0.090972      1      0.30162
       7    Six_Bars          Six Bars          0.091003      1      0.30167
                                                             ==
                                                              7
```

In the iteration history, an asterisk is used to indicate places where *D*-efficiency is greater than any previously reported value.

We see three parameters for brand (4 alternatives including None minus 1), three for price $(4-1)$, one for count $(2-1)$. All are estimable, and all have reasonable standard errors. The variances for the brand effects are the inverse of the number of choice sets $(1/16 = 0.06250)$. The other variances are bigger. These results look good.

One thing to note is that the *D*-efficiency is 12.9142, which is the same as we saw when we used a candidate set of alternatives. This might be surprising for two reasons. First, this design is restricted but the previous design was not. Second, this design was created from a candidate set of choice sets rather than a candidate set of alternatives. The latter approach usually provides more freedom than the former and with a smaller candidate set. If you look at the design from the previous example on page 171, you will see that it conforms to our restrictions even though they were not formally imposed. Designs without constant attributes within choice sets tend to be more efficient than designs with constant attributes. Hence, in this case, the restrictions did not have any effect. However, paring down the candidate set by eliminating less-than-optimal candidates made it easier for the `%ChoicEff` macro to find a good design. Still, even the full-factorial at 512 choice sets is small enough that the `%ChoicEff` macro has no trouble searching it, particularly with as few attributes and alternatives as are in this problem.

The following step displays the first four choice sets:

```
proc print data=best(obs=16);
   by notsorted set;
   id set;
   var brand -- count;
   run;
```

The first four choice sets are as follows:

```
                              Cereal Bars

              Set    Brand           Price      Count

              140    Branolicious    $2.99    Eight Bars
                     Brantopia       $2.89    Six Bars
                     Brantasia       $3.09    Six Bars
                     None

              115    Branolicious    $2.99    Six Bars
                     Brantopia       $3.09    Six Bars
                     Brantasia       $3.19    Eight Bars
                     None

               19    Branolicious    $2.89    Six Bars
                     Brantopia       $2.99    Eight Bars
                     Brantasia       $3.09    Eight Bars
                     None

              164    Branolicious    $2.99    Eight Bars
                     Brantopia       $3.09    Six Bars
                     Brantasia       $3.19    Eight Bars
                     None
```

With this approach, the choice set number refers to the original choice set numbers in the candidate set. Usually, they are in a random order. You can assign consecutive choice set numbers as follows:

```
data sasuser.choice;
   set best(keep=brand price count);
   retain Set 1;
   output;
   if brand = 'None' then set + 1;
   run;

proc print data=sasuser.choice(obs=16);
   by set;
   id set;
   var brand -- count;
   run;
```

The DATA step also stores the final design in a permanent SAS data set so that it is around at analysis time.

The results are as follows:

```
                               Cereal Bars

                 Set    Brand           Price      Count

                  1     Branolicious    $2.99    Eight Bars
                        Brantopia       $2.89    Six Bars
                        Brantasia       $3.09    Six Bars
                        None

                  2     Branolicious    $2.99    Six Bars
                        Brantopia       $3.09    Six Bars
                        Brantasia       $3.19    Eight Bars
                        None

                  3     Branolicious    $2.89    Six Bars
                        Brantopia       $2.99    Eight Bars
                        Brantasia       $3.09    Eight Bars
                        None

                  4     Branolicious    $2.99    Eight Bars
                        Brantopia       $3.09    Six Bars
                        Brantasia       $3.19    Eight Bars
                        None
```

It is instructive to look at the covariance matrix of the parameter estimates. The following steps display this matrix, which is automatically output to a SAS data set called `bestcov`:

```
proc format;
   value zer -1e-12 - 1e-12 = ' 0   ';
   run;

proc print data=bestcov label;
   id __label;
   label __label = '00'x;
   var Branolicious -- Six_Bars;
   format _numeric_ zer5.2;
   run;
```

The results are as follows:

```
                                  Cereal Bars

                                                                       Six
                 Branolicious  Brantasia  Brantopia  $2.89  $2.99  $3.09  Bars

    Branolicious     0.06          0          0        0      0      0      0
    Brantasia        0             0.06       0        0      0      0      0
    Brantopia        0             0          0.06     0      0      0      0
    $2.89            0             0          0        0.09  -0.00  -0.00   0.00
    $2.99            0             0          0       -0.00   0.09   0.00  -0.00
    $3.09            0             0          0       -0.00   0.00   0.09  -0.00
    Six Bars         0             0          0        0.00  -0.00  -0.00   0.09
```

Like before, there are some nonzero covariances between the price and count attributes.

There is one more test that should be run before a design is used. In the following step, the `%MktDups` macro checks the design to see if any choice sets are duplicates of any other choice sets:

```
   %mktdups(branded,                        /* a design with brands          */
            data=sasuser.choice,            /* the input design to evaluate  */
            factors=brand price count,      /* factors in the design         */
            nalts=4)                        /* number of alternatives        */
```

The results are as follows:

```
   Design:          Branded
   Factors:         brand price count
                    Brand
                    Count Price
   Duplicate Sets:  0
```

There are no duplicate choice sets. Collecting and analyzing data is no different than was illustrated previously, so it will not be shown here.

Example 6, A Generic Choice Design

In this example, we create a design for a purely generic experiment—an experiment involving no brands, just bundles of attributes. The design is efficient for a choice model under the null hypothesis $\boldsymbol{\beta} = \mathbf{0}$ and a main-effects model. The design is constructed from a candidate set of alternatives using the `%ChoicEff` macro. Imagine that the manufacturer is interested in better understanding choices for cereal bars independent of brand. Like before, they are interested in price and count, but now they are also interested in the number of calories and whether consumers are influenced by claims such as “naturally cholesterol free.”

Design Factors and Levels

Factor Name	Attribute	Number of Levels	Levels
x1	Price	4	$2.89, $2.99, $3.09, $3.19
x2	Count	2	Six, Eight
x3	Calories	3	90, 110, 130
x4	Cholesterol	2	Cholesterol Free, No Claim

For this experiment, we will use the `%MktEx` macro to make a candidate set of alternatives and then use the `%ChoicEff` macro to create a choice design from the candidate alternatives. First, we can use the `%MktRuns` macro to suggest sizes for the candidate set. We have four factors since there are four attributes with 4, 2, 3, and 2 levels. We use the `%MktRuns` macro as follows:

```
%mktruns(4 2 3 2)                 /* factor level list for one alternative */
```

The results are as follows:

```
                              Cereal Bars

                            Design Summary

                        Number of
                        Levels       Frequency

                             2           2
                             3           1
                             4           1

                              Cereal Bars

          Saturated      = 8
          Full Factorial = 48

          Some Reasonable                       Cannot Be
             Design Sizes       Violations      Divided By

                       24 *              0
                       48 *              0
                       12                2      8
                       36                2      8
                        8 S              4      3  6 12
                       16                4      3  6 12
                       32                4      3  6 12
                       40                4      3  6 12
                       18                5      4  8 12
                       30                5      4  8 12

     * - 100% Efficient design can be made with the MktEx macro.
     S - Saturated Design - The smallest design that can be made.
```

```
                              Cereal Bars

              n              Design              Reference

             24    2 ** 13  3 **  1   4 **  1    Orthogonal Array
             48    2 ** 37  3 **  1   4 **  1    Orthogonal Array
             48    2 ** 34  3 **  1   4 **  2    Orthogonal Array
             48    2 ** 31  3 **  1   4 **  3    Orthogonal Array
             48    2 ** 28  3 **  1   4 **  4    Orthogonal Array
             48    2 ** 25  3 **  1   4 **  5    Orthogonal Array
             48    2 ** 22  3 **  1   4 **  6    Orthogonal Array
             48    2 ** 19  3 **  1   4 **  7    Orthogonal Array
             48    2 ** 16  3 **  1   4 **  8    Orthogonal Array
             48    2 ** 13  3 **  1   4 **  9    Orthogonal Array
             48    2 ** 10  3 **  1   4 ** 10    Orthogonal Array
             48    2 **  7  3 **  1   4 ** 11    Orthogonal Array
             48    2 **  4  3 **  1   4 ** 12    Orthogonal Array
```

This approach results in much smaller designs compared to the linear arrangement approach since you are only creating factors for one alternative at a time instead of factors for all of the attributes of all of the alternatives. The `%MktRuns` macro lists orthogonal arrays that the `%MktEx` macro knows how to make. In some cases, these might provide good candidate designs. However, in this case, 48 runs is sufficiently small that we can use the full-factorial design as a candidate set. The `%MktEx` macro can be used as follows to make the candidate alternatives:

```
   %mktex(4 2 3 2,            /* factor level list for one alternative       */
          n=48)               /* number of candidate alternatives            */
```

The results are as follows:

```
                              Cereal Bars

                        Algorithm Search History

                            Current         Best
        Design    Row,Col  D-Efficiency  D-Efficiency  Notes
        ----------------------------------------------------------
             1      Start      100.0000      100.0000  Tab
             1        End      100.0000
```

```
                                Cereal Bars

                           The OPTEX Procedure

                         Class Level Information

                          Class  Levels  Values

                          x1       4     1 2 3 4
                          x2       2     1 2
                          x3       3     1 2 3
                          x4       2     1 2
                                Cereal Bars

                                                                Average
                                                               Prediction
Design                                                          Standard
Number     D-Efficiency     A-Efficiency     G-Efficiency        Error
------------------------------------------------------------------------
     1       100.0000         100.0000         100.0000          0.4082
```

The full-factorial design has 100% *D*-efficiency. This design, like all designs that come out of the `%MktEx` macro, has factor names `x1`, `x2`, and so on, and levels of 1, 2, and so on. The next steps do several things. The first step specifies formats for the levels of the attributes. The `%MktLab` macro step assigns meaningful variable names and the formats. In this design, we will have three alternatives, so the `%MktLab` macro creates three new variables, `f1-f3` with the `int=` or intercept option. The values of these three variables are all ones. They are used as flags to indicate that every candidate can appear in every alternative. The following steps create the candidate design:

```
proc format;
   value price 1 = $2.89 2 = $2.99 3 = $3.09 4 = $3.19;
   value count 1 = 'Six Bars' 2 = 'Eight Bars';
   value cal   1 = '90 Calories' 2 = '110 Calories' 3 = '130 Calories';
   value chol  1 = 'Cholesterol Free' 2 = 'No Claim';
   run;

%mktlab(data=design,           /* input data set                         */
        vars=Price Count Calories Cholesterol, /* new attribute names       */
        int=f1-f3,             /* create 3 columns of 1's in f1-f3       */
        out=final,             /* output design                          */
                               /* add a format statement for the attributes */
        stmts=format price price. count count. calories cal. cholesterol chol.)
```

The following step displays the candidates:

```
proc print; run;
```

The results are as follows:

```
                                Cereal Bars

 Obs    f1    f2    f3    Price      Count          Calories       Cholesterol

   1     1     1     1    $2.89    Six Bars       90 Calories     Cholesterol Free
   2     1     1     1    $2.89    Six Bars       90 Calories     No Claim
   3     1     1     1    $2.89    Six Bars       110 Calories    Cholesterol Free
   4     1     1     1    $2.89    Six Bars       110 Calories    No Claim
   5     1     1     1    $2.89    Six Bars       130 Calories    Cholesterol Free
   6     1     1     1    $2.89    Six Bars       130 Calories    No Claim
   7     1     1     1    $2.89    Eight Bars     90 Calories     Cholesterol Free
   8     1     1     1    $2.89    Eight Bars     90 Calories     No Claim
   9     1     1     1    $2.89    Eight Bars     110 Calories    Cholesterol Free
  10     1     1     1    $2.89    Eight Bars     110 Calories    No Claim
  11     1     1     1    $2.89    Eight Bars     130 Calories    Cholesterol Free
  12     1     1     1    $2.89    Eight Bars     130 Calories    No Claim
  13     1     1     1    $2.99    Six Bars       90 Calories     Cholesterol Free
  14     1     1     1    $2.99    Six Bars       90 Calories     No Claim
  15     1     1     1    $2.99    Six Bars       110 Calories    Cholesterol Free
  16     1     1     1    $2.99    Six Bars       110 Calories    No Claim
  17     1     1     1    $2.99    Six Bars       130 Calories    Cholesterol Free
  18     1     1     1    $2.99    Six Bars       130 Calories    No Claim
  19     1     1     1    $2.99    Eight Bars     90 Calories     Cholesterol Free
  20     1     1     1    $2.99    Eight Bars     90 Calories     No Claim
  21     1     1     1    $2.99    Eight Bars     110 Calories    Cholesterol Free
  22     1     1     1    $2.99    Eight Bars     110 Calories    No Claim
  23     1     1     1    $2.99    Eight Bars     130 Calories    Cholesterol Free
  24     1     1     1    $2.99    Eight Bars     130 Calories    No Claim
  25     1     1     1    $3.09    Six Bars       90 Calories     Cholesterol Free
  26     1     1     1    $3.09    Six Bars       90 Calories     No Claim
  27     1     1     1    $3.09    Six Bars       110 Calories    Cholesterol Free
  28     1     1     1    $3.09    Six Bars       110 Calories    No Claim
  29     1     1     1    $3.09    Six Bars       130 Calories    Cholesterol Free
  30     1     1     1    $3.09    Six Bars       130 Calories    No Claim
  31     1     1     1    $3.09    Eight Bars     90 Calories     Cholesterol Free
  32     1     1     1    $3.09    Eight Bars     90 Calories     No Claim
  33     1     1     1    $3.09    Eight Bars     110 Calories    Cholesterol Free
  34     1     1     1    $3.09    Eight Bars     110 Calories    No Claim
  35     1     1     1    $3.09    Eight Bars     130 Calories    Cholesterol Free
  36     1     1     1    $3.09    Eight Bars     130 Calories    No Claim
  37     1     1     1    $3.19    Six Bars       90 Calories     Cholesterol Free
  38     1     1     1    $3.19    Six Bars       90 Calories     No Claim
  39     1     1     1    $3.19    Six Bars       110 Calories    Cholesterol Free
  40     1     1     1    $3.19    Six Bars       110 Calories    No Claim
```

```
   41      1      1      1    $3.19    Six Bars      130 Calories    Cholesterol Free
   42      1      1      1    $3.19    Six Bars      130 Calories    No Claim
   43      1      1      1    $3.19    Eight Bars    90 Calories     Cholesterol Free
   44      1      1      1    $3.19    Eight Bars    90 Calories     No Claim

   45      1      1      1    $3.19    Eight Bars    110 Calories    Cholesterol Free
   46      1      1      1    $3.19    Eight Bars    110 Calories    No Claim
   47      1      1      1    $3.19    Eight Bars    130 Calories    Cholesterol Free
   48      1      1      1    $3.19    Eight Bars    130 Calories    No Claim
```

The candidate design is structured so that every candidate alternative can appear anywhere in the final design. You can see this by looking at the flag variables, `f1-f3`. When `f1` = 1, then the candidate can be used in the first alternative; when `f2` = 1, then the candidate can be used in the second alternative; and when `f3` = 1, then the candidate can be used in the third alternative. This candidate design is then input to the `%ChoicEff` macro as follows:

```
%choiceff(data=final,                 /* candidate set of alternatives      */
          bestout=sasuser.cerealdes,  /* choice design permanently stored   */
                                      /* model with stdz orthog coding      */
          model=class(price count calories cholesterol / sta) /
                cprefix=0             /* lpr=0 labels from just levels      */
                lprefix=0,            /* cpr=0 names from just levels       */
          nsets=9,                    /* number of choice sets to make      */
          seed=145,                   /* random number seed                 */
          flags=f1-f3,                /* flag which alt can go where, 3 alts*/
          options=relative,           /* display relative D-efficiency      */
          beta=zero)                  /* assumed beta vector                */
```

The `%ChoicEff` step creates a generic choice design with 9 choice sets and 3 alternatives. The model specification specifies a main-effects model and the standardized orthogonal contrast coding. The `cprefix=0` option is specified so that variable names are constructed just from the attribute levels using zero characters of the attribute (or class) variable names. Similarly, the `lprefix=0` option is specified so that variable labels are constructed just from the attribute levels using zero characters of the attribute (or class) variable names or labels. The results are as follows:

```
                                   Cereal Bars

                     n    Name                Beta    Label

                     1    _2_89                 0     $2.89
                     2    _2_99                 0     $2.99
                     3    _3_09                 0     $3.09
                     4    Six_Bars              0     Six Bars
                     5    _90_Calories          0     90 Calories
                     6    _110_Calories         0     110 Calories
                     7    Cholesterol_Free      0     Cholesterol Free
```

```
                    Design   Iteration  D-Efficiency       D-Error
                    ----------------------------------------------
                         1       0                 0             .
                                 1           7.79175 *     0.12834
                                 2           8.15348 *     0.12265
                                 3           8.17776 *     0.12228

                    Design   Iteration  D-Efficiency       D-Error
                    ----------------------------------------------
                         2       0           4.44502       0.22497
                                 1           8.03861       0.12440
                                 2           8.18335 *     0.12220
                                 3           8.20929 *     0.12181

                                   Cereal Bars

                                  Final Results

                              Design                 2
                              Choice Sets            9
                              Alternatives           3
                              Parameters             7
                              Maximum Parameters    18
                              D-Efficiency      8.2093
                              Relative D-Eff   91.2143
                              D-Error           0.1218
                              1 / Choice Sets   0.1111

                                   Cereal Bars

                                                                            Standard
  n    Variable Name       Label                Variance    DF       Error

  1    _2_89               $2.89                 0.13420     1     0.36633
  2    _2_99               $2.99                 0.12509     1     0.35368
  3    _3_09               $3.09                 0.12331     1     0.35115
  4    Six_Bars            Six Bars              0.12539     1     0.35410
  5    _90_Calories        90 Calories           0.11310     1     0.33630
  6    _110_Calories       110 Calories          0.11310     1     0.33630
  7    Cholesterol_Free    Cholesterol Free      0.12622     1     0.35528
                                                                ==
                                                                 7
```

The first table lists the parameters and their assumed values (all zero). The next two tables show the iteration history. The results for iteration 0 are the results for the initial random selection of alternatives. It is often the case that the *D*-efficiency for the initial random design is zero, but with iteration, the efficiency increases. The final two tables provide information about the design specification, the final *D*-efficiency, and the variances and standard errors. We see three parameters for brand (4 alternatives including None minus 1), three for price (4 − 1), one for count (2 − 1). All are estimable, and all have reasonable standard errors. The best you can hope for with three level factors is a variance of $1/9 \approx 0.11111$. This design looks good. The standard errors are all similar and close to the minimum, and all of the parameters can be estimated. The following step displays the design:

```
proc print data=sasuser.cerealdes;
   var price -- cholesterol;
   id set; by set;
   run;
```

The results are as follows:

```
Set    Price      Count         Calories       Cholesterol

 1     $3.19    Eight Bars    90 Calories      No Claim
       $2.89    Six Bars      110 Calories     No Claim
       $3.09    Six Bars      130 Calories     Cholesterol Free

 2     $3.19    Eight Bars    130 Calories     No Claim
       $2.99    Six Bars      90 Calories      Cholesterol Free
       $2.89    Six Bars      110 Calories     Cholesterol Free

 3     $3.19    Six Bars      110 Calories     Cholesterol Free
       $2.89    Eight Bars    130 Calories     No Claim
       $3.09    Eight Bars    90 Calories      Cholesterol Free

 4     $3.19    Eight Bars    110 Calories     Cholesterol Free
       $2.99    Six Bars      130 Calories     No Claim
       $3.09    Six Bars      90 Calories      Cholesterol Free

 5     $2.99    Eight Bars    130 Calories     Cholesterol Free
       $2.89    Six Bars      90 Calories      Cholesterol Free
       $3.09    Eight Bars    110 Calories     No Claim

 6     $2.89    Eight Bars    130 Calories     Cholesterol Free
       $2.99    Eight Bars    110 Calories     Cholesterol Free
       $3.19    Six Bars      90 Calories      No Claim

 7     $3.09    Eight Bars    130 Calories     Cholesterol Free
       $2.89    Eight Bars    90 Calories      No Claim
       $2.99    Six Bars      110 Calories     No Claim

 8     $3.19    Six Bars      130 Calories     Cholesterol Free
       $2.99    Eight Bars    90 Calories      No Claim
       $3.09    Eight Bars    110 Calories     No Claim

 9     $2.99    Eight Bars    90 Calories      Cholesterol Free
       $3.09    Six Bars      130 Calories     No Claim
       $2.89    Eight Bars    110 Calories     Cholesterol Free
```

The following step is not necessary, but it is instructive to look at the covariance matrix of the parameter estimates. The following steps display this matrix, which is automatically output to a SAS data set called `bestcov`:

```
proc format;
   value zer -1e-12 - 1e-12 = ' 0   ';
   run;

proc print data=bestcov label;
   id __label;
   label __label = '00'x;
   var _2_89 -- Cholesterol_Free;
   format _numeric_ zer5.2;
   run;
```

The results are as follows:

```
                                      Cereal Bars

                                             Six       90       110     Cholesterol
                         $2.89  $2.99  $3.09  Bars   Calories  Calories     Free

     $2.89                0.13   0.01   0.01  0.00     0.00     -0.01       -0.01
     $2.99                0.01   0.13   0.00  0.00    -0.01      0.01       -0.00
     $3.09                0.01   0.00   0.12  0.00     0.01      0.01       -0.01
     Six Bars             0.00   0.00   0.00  0.13     0.00      0.00       -0.01
     90 Calories          0.00  -0.01   0.01  0.00     0.11      0.00       -0.00
     110 Calories        -0.01   0.01   0.01  0.00     0.00      0.11       -0.00
     Cholesterol Free    -0.01  -0.00  -0.01 -0.01    -0.00     -0.00        0.13
```

There are some nonzero covariances between the price and count attributes.

There is one more test that should be run before a design is used. In the following step, the `%MktDups` macro checks the design to see if any choice sets are duplicates of any other choice sets:

```
%mktdups(generic,                     /* generic design (no brands)          */
         data=sasuser.cerealdes,      /* the input design to evaluate        */
                                      /* factors in the design               */
         factors=price count calories cholesterol,
         nalts=3)                     /* number of alternatives              */
```

The results are as follows:

```
Design:          Generic
Factors:         price count calories cholesterol
                 Calories Cholesterol Count Price
Sets w Dup Alts: 0
Duplicate Sets:  0
```

The results contain a line that we did not see with branded designs. With generic designs, the macro also checks for duplicate alternatives within choice sets. In this case, there are no duplicate choice sets and no duplicate alternatives within choice sets.

Collecting and analyzing data is no different than was illustrated previously, so it will not be shown here.

Example 7, A Partial-Profile Choice Experiment

This example is like the previous example in the sense that we will create a design for a generic experiment—an experiment involving no brands, just bundles of attributes. We will create a design that is optimal for a main-effects choice model under the null hypothesis $\boldsymbol{\beta} = \mathbf{0}$. We will continue to work with cereal bars. This time, the manufacturer is interested in knowing people's preferences for ingredients of high-end bars. The goal is to construct an experiment with 13 attributes, all described by the presence or absence of the following ingredients:

Almonds
Apple
Banana Chips
Brown Sugar
Cashews
Chocolate
Coconut
Cranberries
Hazel Nuts
Peanuts
Pecans
Raisins
Walnuts

Since it might be difficult for people to compare that many ingredients simultaneously, we will only vary subsets of the attributes at any one time. This kind of experiment is called a partial-profile choice experiment (Chrzan and Elrod 1995), and the experimental design is made from a balanced incomplete block design (BIBD) and an orthogonal array. See page 115 for more about BIBDs. See page 58 for more about orthogonal arrays. We need to construct a design where our $t = 13$ attributes can be shown in b blocks of choice sets where k attributes vary in each block. The next task is to determine values for b and k. We can use the `%MktBSize` macro for this. It tells us sizes in which a BIBD or unbalanced block design might be possible. The following step sets the number of attributes to 13 and asks for sizes in the default range of 2 to 500 blocks of choice sets with a size in the range 3 to 8:

```
title 'Cereal Bars';

%mktbsize(nattrs=13,                /* 13 attributes                         */
          setsize=3 to 8,           /* try set sizes in range 3 to 8         */
          options=ubd)              /* consider unbalanced designs too       */
```

The results can contain BIBDs (λ, the pairwise frequency is an integer) and also unbalanced block designs (since `options=ubd` was specified) where λ is not an integer. The results are as follows:

```
                            Cereal Bars

                                                                 n
        t            k      b          r             Lambda      Total
        Number of    Set    Number     Attribute     Pairwise    Sample
        Attributes   Size   of Sets    Frequency     Frequencies Size

            13         3       13          3             0.5        39
            13         4       13          4               1        52
            13         5       13          5            1.67        65
            13         6       13          6             2.5        78
            13         7       13          7             3.5        91
            13         8       13          8            4.67       104
```

A BIBD might be possible with 13 blocks of 13 attributes shown 4 at a time. Four seems like a good value to pick for the set size for several reasons. Four seems good because it will work well with an orthogonal array that we could use. The array $4^1 2^4$ in 8 runs can provide the two-level factors that we need to make the partial-profile design. The orthogonal array must be a p^k subset of an array $p^k s^1$ in $p \times s$ runs with $k \leq s$. See page 1145 for more information. Now, returning to the block design, 4 is also good because that leads to a BIBD ($\lambda = 1$, an integer) which is preferable to an unbalanced block design (λ not an integer). However, in fact, four is not good, because it leads to undesirable choice sets (not statistically inefficient, but undesirable for other reasons). To better understand this, let's consider the following potential choice sets with $k = 4$:

```
    Set   x1   x2   x3   x4   x5   x6   x7   x8   x9   x10   x11   x12   x13

     1     .    1    2    .    .    .    1    .    2    .     .     .     .
           .    2    1    .    .    .    2    .    1    .     .     .     .

     2     .    1    1    .    .    .    1    .    1    .     .     .     .
           .    2    2    .    .    .    2    .    2    .     .     .     .

     3     .    2    1    .    .    .    1    .    1    .     .     .     .
           .    1    2    .    .    .    2    .    2    .     .     .     .

     4     .    1    1    .    .    .    1    .    2    .     .     .     .
           .    2    2    .    .    .    2    .    1    .     .     .     .
```

For clarity, these are displayed with 2 for present, 1 for absent, and missing (.) for not varied in this set. Choice set one looks good. The trade off is between attributes 3 and 9 versus 2 and 7 (banana chips and hazel nuts versus apple and coconut). Unfortunately, whenever this kind of choice set appears (two attributes versus two attributes) another type of choice set will also appear, such as the one shown in choice set 2. It pairs four attributes present versus none present. This in no way diminishes statistical design optimality, but it might diminish realism. Do you really want a series of choice sets comparing a plain bar with one with lots of extras? Perhaps; perhaps not. The alternative with $k = 4$ is the kind of choice set shown in sets 3 and 4. They compare one attribute with three others. You will get either a bunch of sets like 1 and 2 or a bunch like 3 and 4 depending on the random number seed you use. In general, there is not going to be a way for you to know how things will work out with your design until you try a particular combination and then carefully evaluate your design to see if it looks okay.

We will try something different. We will try $k = 6$ here and see how that works. Setting $k = 6$ has the potential to lead to comparisons of 3 attributes with 3 other attributes, 2 with 4, and sometimes perhaps 1 with 5. We can create the design as follows:

```
%mktbibd(nattrs=13,                /* 13 attributes                       */
         setsize=6,                /* vary 6 at a time                    */
         b=13,                     /* create 13 blocks of choice sets     */
         seed=289)                 /* random number seed                  */
```

The results are as follows:

```
                                   Cereal Bars

                    Block Design Efficiency Criterion       99.8492
                    Number of Attributes, t                      13
                    Set Size, k                                   6
                    Number of Sets, b                            13
                    Average Attribute Frequency                   6
                    Average Pairwise Frequency                  2.5
                    Total Sample Size                            78
                    Positional Frequencies Optimized?           Yes

                           Attribute by Attribute Frequencies

                           1  2  3  4  5  6  7  8  9 10 11 12 13

                       1   6  3  3  3  3  2  2  2  3  2  3  2  2
                       2      6  3  2  2  2  2  3  2  3  2  3  3
                       3         6  3  2  3  3  2  2  2  2  2  3
                       4            6  3  2  3  3  2  2  2  3  2
                       5               6  2  2  2  3  3  3  2  3
                       6                  6  2  3  3  3  3  2  3
                       7                     6  2  3  3  3  3  2
                       8                        6  3  3  3  2  2
                       9                           6  2  2  3  2
                      10                              6  2  3  2
                      11                                 6  2  3
                      12                                    6  3
                      13                                       6
                    Attribute by Position Frequencies

                             1  2  3  4  5  6

                         1   1  1  1  1  1  1
                         2   1  1  1  1  1  1
                         3   1  1  1  1  1  1
                         4   1  1  1  1  1  1
                         5   1  1  1  1  1  1
                         6   1  1  1  1  1  1
                         7   1  1  1  1  1  1
                         8   1  1  1  1  1  1
                         9   1  1  1  1  1  1
                        10   1  1  1  1  1  1
                        11   1  1  1  1  1  1
                        12   1  1  1  1  1  1
                        13   1  1  1  1  1  1

                             Cereal Bars

                                Design

                   x1    x2    x3    x4    x5    x6

                    1    10     5    13    11     6
                   13     2     3    12    10     5
                    9     1    11     8     5     2
                   12     5    13     4     9     7
                    2     7     9    10    12     1
                   10    11     4     5     7     8
                    8     4    12     6     2    10
                    3     6     7     2    13    11
                    5     9     6     3     1     4
                    6     3    10     7     8     9
                   11    13     8     9     6    12
                    7    12     1    11     4     3
                    4     8     2     1     3    13
```

The less than 100% efficiency shows that a BIBD was not found, as do the nonconstant attribute by attribute frequencies (2's and 3's). However, the constant attribute frequencies (6) show that an unbalanced block design was found. The attribute by position frequencies are perfect. Every attribute appears in every position exactly once. Note, however, that they are *not* important for partial-profile designs (they are important for MaxDiff designs). You can make the macro run faster by specifying `positer=0` so that it will not try to optimize positional frequencies. You can also make it run faster by asking for fewer PROC OPTEX iterations. By default, the macro is trying to find multiple optimal designs so it can pick the one that is best in terms of position frequencies. We can run the macro again as follows to find a design much faster:

```
%mktbibd(nattrs=13,                /* 13 attributes                           */
         setsize=6,                /* vary 6 at a time                        */
         b=13,                     /* create 13 blocks of choice sets         */
         seed=289,                 /* random number seed                      */
         positer=0,                /* do not optimize position frequencies */
         optiter=100)              /* only 100 PROC OPTEX iterations          */
```

The results are as follows:

```
                                 Cereal Bars

                   Block Design Efficiency Criterion       99.8492
                   Number of Attributes, t                      13
                   Set Size, k                                   6
                   Number of Sets, b                            13
                   Average Attribute Frequency                   6
                   Average Pairwise Frequency                  2.5
                   Total Sample Size                            78
                   Positional Frequencies Optimized?            No

                          Attribute by Attribute Frequencies

                          1  2  3  4  5  6  7  8  9 10 11 12 13

                      1   6  3  3  2  3  3  3  2  3  2  2  2  2
                      2      6  2  2  2  3  2  3  2  3  2  3  3
                      3         6  2  3  2  3  3  2  3  2  3  2
                      4            6  2  2  3  3  3  3  2  3  3
                      5               6  2  2  3  2  3  3  2  3
                      6                  6  3  3  2  2  3  3  2
                      7                     6  2  2  2  3  2  3
                      8                        6  3  2  2  2  2
                      9                           6  3  3  3  2
                     10                              6  3  2  2
                     11                                 6  2  3
                     12                                    6  3
                     13                                       6
```

```
                Attribute by Position Frequencies

                          1   2   3   4   5   6

                      1   6   0   0   0   0   0
                      2   3   3   0   0   0   0
                      3   3   1   2   0   0   0
                      4   1   4   1   0   0   0
                      5   0   3   2   1   0   0
                      6   0   2   2   2   0   0
                      7   0   0   4   0   2   0
                      8   0   0   2   2   1   1
                      9   0   0   0   5   1   0
                     10   0   0   0   2   3   1
                     11   0   0   0   1   3   2
                     12   0   0   0   0   3   3
                     13   0   0   0   0   0   6

                          Cereal Bars

                             Design

                x1    x2    x3    x4    x5    x6

                 3     5     6     8    10    11
                 3     4     7     8     9    10
                 2     4     6    10    12    13
                 1     2     4     6     7     8
                 1     2     3     9    10    12
                 1     3     5     6     7    12
                 1     4     5     9    10    11
                 2     6     8     9    11    12
                 3     4     7    11    12    13
                 1     6     7     9    11    13
                 4     5     8     9    12    13
                 2     5     7    10    11    13
                 1     2     3     5     8    13
```

Our efficiency is the same, and this step ran on the order of a few seconds (compared to on the order of a minute for the previous step). The first row of the design specifies that in the first block of choice sets, attributes 2, 5, 6, 8, 10, and 11 will vary (Apple, Cashews, Chocolate, Cranberries, Peanuts, and Pecans) while the others stay constant.

Next, we will need an orthogonal array with 6 two-level attributes. The next biggest power of 2 (because we have two-level factors) greater than 6 (because we have 6 of them) is 8. We will need to divide our orthogonal array into two blocks of size 8. More is said about this later. Also see page 1145.

The following steps make the orthogonal array, combine it with the BIBD using the `%MktPPro` macro, and evaluate the resulting choice design:

```
%mktex(8 2 ** 6,                   /* 1 eight-level and 6 two-level factors*/
       n=16,                       /* 16 runs                              */
       seed=382)                   /* random number seed                   */

proc sort data=randomized          /* sort randomized data set             */
          out=randes(drop=x1);     /* do not need 8-level factor any more  */
   by x2 x1;                       /* must sort by x2 then x1.  Really!    */
   run;

%mktppro(ibd=bibd,                 /* input block design                   */
         design=randes)            /* input orthogonal array               */

%choiceff(data=chdes,              /* candidate set of choice sets         */
          init=chdes,              /* initial design                       */
          initvars=x1-x13,         /* factors in the initial design        */
          model=class(x1-x13 / sta),/* model with stdz orthogonal coding   */
          nsets=104,               /* number of choice sets                */
          nalts=2,                 /* number of alternatives               */
          rscale=                  /* relative D-efficiency scale factor   */
          %sysevalf(104 * 6 / 13), /* 6 of 13 attrs in 104 sets vary       */
          beta=zero)               /* assumed beta vector, Ho: b=0         */
```

The design has 104 choice sets ($b = 13$ times $s = 8$). The top half ($s = 8$) rows of the orthogonal array form the first alternatives, and the second half form the second. The last part of the output from the `%ChoicEff` macro is as follows:

```
                              Cereal Bars

                             Final Results

                        Design                  1
                        Choice Sets           104
                        Alternatives            2
                        Parameters             13
                        Maximum Parameters    104
                        D-Efficiency      48.0000
                        Relative D-Eff   100.0000
                        D-Error            0.0208
                        1 / Choice Sets  0.009615
```

```
                                  Cereal Bars

                   Variable                                        Standard
              n      Name      Label     Variance    DF              Error

              1      x11       x1 1      0.020833     1            0.14434
              2      x21       x2 1      0.020833     1            0.14434
              3      x31       x3 1      0.020833     1            0.14434
              4      x41       x4 1      0.020833     1            0.14434
              5      x51       x5 1      0.020833     1            0.14434
              6      x61       x6 1      0.020833     1            0.14434
              7      x71       x7 1      0.020833     1            0.14434
              8      x81       x8 1      0.020833     1            0.14434
              9      x91       x9 1      0.020833     1            0.14434
             10      x101      x10 1     0.020833     1            0.14434
             11      x111      x11 1     0.020833     1            0.14434
             12      x121      x12 1     0.020833     1            0.14434
             13      x131      x13 1     0.020833     1            0.14434
                                                     ==
                                                     13
```

The macro reports a relative *D*-efficiency of 100. If we had not specified the `rscale=` option, the relative *D*-efficiency would have been 46.1538. This should be compared to the maximum possible relative *D*-efficiency, which is $100(k/t) = 100(6/13) = 46.1538$. The design is (6 / 13)*th* as efficient as a generic design with all 13 attributes simultaneously varying, so it is optimal for this partial-profile experiment. When there are no restrictions, you expect *D*-efficiency to equal the number of choice sets. We are getting (6 / 13)*th* as much information as that. Hence, for an optimal partial-profile design with 6 of 13 attributes varying in 104 choice sets, we expect a *D*-efficiency of (6 / 13)*th* of 104. Specifying that value in the `rscale=` option gives us a relative *D*-efficiency that is relative to an optimal partial-profile design, which is in fact what we have. Note that the expression `%sysevalf(104 * 6 / 13)` is evaluated by the macro processor before the macro is invoked, and the resulting number is passed to the `%ChoicEff` macro. Since the result is not an integer, we cannot use the `%eval` function.

The first 16 choice sets are displayed as follows:

```
   proc print data=chdes; id set; by set; where set le 16; run;
```

The results are as follows:

```
                                  Cereal Bars

     Set   x1   x2   x3   x4   x5   x6   x7   x8   x9   x10   x11   x12   x13

       1    1    1    1    1    2    2    1    2    1    2     2     1     1
            1    1    2    1    1    1    1    1    1    1     1     1     1

       2    1    1    1    1    1    1    1    2    1    1     2     1     1
            1    1    2    1    2    2    1    1    1    2     1     1     1
```

```
  3    1    1    1    1    1    1    1    2    1    2     1     1     1
       1    1    2    1    2    2    1    1    1    1     2     1     1

  4    1    1    1    1    1    2    1    1    1    2     1     1     1
       1    1    2    1    2    1    1    2    1    1     2     1     1

  5    1    1    1    1    2    1    1    1    1    2     2     1     1
       1    1    2    1    1    2    1    2    1    1     1     1     1

  6    1    1    1    1    1    2    1    1    1    1     2     1     1
       1    1    2    1    2    1    1    2    1    2     1     1     1

  7    1    1    1    1    2    2    1    2    1    1     1     1     1
       1    1    2    1    1    1    1    1    1    2     2     1     1

  8    1    1    1    1    2    1    1    1    1    1     1     1     1
       1    1    2    1    1    2    1    2    1    2     2     1     1

  9    1    1    1    2    1    1    2    2    2    2     1     1     1
       1    1    2    1    1    1    1    1    1    1     1     1     1

 10    1    1    1    1    1    1    1    2    1    2     1     1     1
       1    1    2    2    1    1    2    1    2    1     1     1     1

 11    1    1    1    1    1    1    1    2    2    1     1     1     1
       1    1    2    2    1    1    2    1    1    2     1     1     1

 12    1    1    1    1    1    1    2    1    2    1     1     1     1
       1    1    2    2    1    1    1    2    1    2     1     1     1

 13    1    1    1    2    1    1    1    1    2    2     1     1     1
       1    1    2    1    1    1    2    2    1    1     1     1     1

 14    1    1    1    1    1    1    2    1    1    2     1     1     1
       1    1    2    2    1    1    1    2    2    1     1     1     1

 15    1    1    1    2    1    1    2    2    1    1     1     1     1
       1    1    2    1    1    1    1    1    2    2     1     1     1

 16    1    1    1    2    1    1    1    1    1    1     1     1     1
       1    1    2    1    1    1    2    2    2    2     1     1     1
```

When an attribute is all 1's in a set, then that attribute does not vary. Exactly six attributes vary in each set. The first 8 choice sets are constructed from the orthogonal array and the first row of the BIBD, the next 8 choice sets are constructed from the orthogonal array and the second row of the BIBD, and so on. In the first 8 choice sets, attributes 3, 5, 6, 8, 10 and 11 vary, and in the second block, it is 3, 4, 7, 8, 9 and 10.

You can display the orthogonal array as follows:

```
proc print noobs data=randes; run;
```

The results are as follows:

```
                         Cereal Bars

               x2    x3    x4    x5    x6    x7

                1     2     2     2     2     2
                1     1     1     2     1     2
                1     1     1     2     2     1
                1     1     2     1     2     1
                1     2     1     1     2     2
                1     1     2     1     1     2
                1     2     2     2     1     1
                1     2     1     1     1     1
                2     1     1     1     1     1
                2     2     2     1     2     1
                2     2     2     1     1     2
                2     2     1     2     1     2
                2     1     2     2     1     1
                2     2     1     2     2     1
                2     1     1     1     2     2
                2     1     2     2     2     2
```

This matrix has two blocks corresponding to `x2` = 1 and `x2` = 2. These are called difference schemes (although the full difference schemes have 8 columns not just these 6). Understanding this is not critical, but you can go to page 115 for more information. What is important to understand is that the seemingly odd sorting of the randomized design by `x2` and then by `x1` and the dropping of `x1` is required. It guarantees that the orthogonal array has this layout of stacked difference schemes. The randomized design is used since it is much less likely to consist of rows that are constant (usually all ones) than the original design that is stored by default in the `out=design` data set. Orthogonal arrays that can be used to make optimal partial-profile designs include: $2^4 4^1$ in 8 runs, $2^8 8^1$ in 16 runs, 4^5 in 16 runs, $3^6 6^1$ in 18 runs, $2^{12} 12^1$ in 24 runs, 5^6 in 25 runs, $3^9 9^1$ in 27 runs, $2^{16} 16^1$ in 32 runs, $4^8 8^1$ in 32 runs, $3^{12} 12^1$ in 36 runs, $2^{20} 20^1$ in 40 runs, $3^9 15^1$ in 45 runs, $2^{24} 24^1$ in 48 runs, $4^{12} 12^1$ in 48 runs, 7^8 in 49 runs, $5^{10} 10^1$ in 50 runs, $3^{18} 18^1$ in 54 runs, $2^{28} 28^1$ in 56 runs, $3^{12} 21^1$ in 63 runs, $2^{32} 32^1$ in 64 runs, $4^{16} 16^1$ in 64 runs, and so on.

Next, the attributes that vary from the first four choice sets are displayed:

```
                              Cereal Bars

                       Set   x3   x5   x6   x8   x10  x11

                         1    1    2    2    2    2    2
                              2    1    1    1    1    1
                         2    1    1    1    2    1    2
                              2    2    2    1    2    1
                         3    1    1    1    2    2    1
                              2    2    2    1    1    2
                         4    1    1    2    1    2    1
                              2    2    1    2    1    2
                         5    1    2    1    1    2    2
                              2    1    2    2    1    1
                         6    1    1    2    1    1    2
                              2    2    1    2    2    1
                         7    1    2    2    2    1    1
                              2    1    1    1    2    2
                         8    1    2    1    1    1    1
                              2    1    2    2    2    2
```

Each set consists of one row from the top block of the orthogonal array and its corresponding row from the bottom block, stored in the locations dictated by the BIBD.

At 104 choice sets, most researchers would consider this design to be too large for one person to evaluate, so it could be blocked into 8 subdesigns of size 13 as follows:

```
%mktblock(data=chdes,                /* input choice design to block        */
          out=sasuser.chdes,         /* output blocked choice design        */
                                     /* stored in permanent SAS data set    */
          nalts=2,                   /* two alternatives                    */
          nblocks=8,                 /* eight blocks                        */
          factors=x1-x13,            /* 13 attributes, x1-x13               */
          print=design,              /* print the blocked design (only)     */
          seed=472)                  /* random number seed                  */
```

A sample of the resulting partial-profile design is as follows:

```
                                Cereal Bars

Block  Set  Alt  x1  x2  x3  x4  x5  x6  x7  x8  x9  x10  x11  x12  x13

  1      1   1    1   1   1   1   1   1   1   2   1   1    2    1    1
             2    1   1   2   1   2   2   1   1   1   2    1    1    1

  .
  .
  .

  1     13   1    1   2   1   1   1   1   1   2   1   1    1    1    2
             2    2   1   2   1   2   1   1   1   1   1    1    1    1

  2      1   1    1   1   1   1   2   2   1   2   1   1    1    1    1
             2    1   1   2   1   1   1   1   1   1   2    2    1    1

  .
  .
  .

  2     13   1    1   1   2   1   1   1   1   2   1   1    1    1    1
             2    2   2   1   1   2   1   1   1   1   1    1    1    2

  3      1   1    1   1   1   1   2   1   1   1   1   2    2    1    1
             2    1   1   2   1   1   2   1   2   1   1    1    1    1

  .
  .
  .

  3     13   1    1   2   1   1   1   1   1   1   1   1    1    1    1
             2    2   1   2   1   2   1   1   2   1   1    1    1    2

  .
  .
  .

  8      1   1    1   1   1   1   1   2   1   1   1   1    2    1    1
             2    1   1   2   1   2   1   1   2   1   2    1    1    1
  .
  .
  .

  8     13   1    1   1   1   1   2   1   1   1   1   1    1    1    2
             2    2   2   2   1   1   1   1   2   1   1    1    1    1
```

The following DATA step displays a summary of the design using the actual attribute levels:

```
data _null_;
   array alts[13] $ 12 _temporary_ ('Almonds' 'Apple' 'Banana Chips'
                  'Brown Sugar' 'Cashews' 'Chocolate' 'Coconut' 'Cranberries'
                  'Hazel Nuts' 'Peanuts' 'Pecans' 'Raisins' 'Walnuts');
   array x[13];                            /* 13 design factors            */
   set sasuser.chdes;                      /* read each alternative        */
   if alt eq 1 then put / block 1. set 3. +1 @; /* write block and set num  */
   else put '<vs> ' @;                     /* print '<vs>' to separate alts  */
   c = 0;                                  /* do not print a comma yet     */
   do j = 1 to 13;                         /* loop over all 13 attrs       */
      if x[j] eq 2 then do;                /* if this one is shown         */
         if c then put +(-1) ', ' @;       /* print comma if not on 1st attr */
         put alts[j] @;                    /* print attr value             */
         c = 1;                            /* not on first term so do commas */
         end;
      end;
   run;
```

The first `array` statement creates a temporary array (no individual variable names are created or stored) of character variables (by virtue of the "$") of length 12 and initializes them to the attribute labels. The `put` statement prints lines, and lines produced by `put` statements that end in "`@`" are held so that the next `put` statement can add to it. The first `put` statement begins with a slash, which clears the previous line and starts a new line. The first block of choice sets is as follows:

```
1  1 Cranberries, Pecans <vs> Banana Chips, Cashews, Chocolate, Peanuts
1  2 Brown Sugar <vs> Banana Chips, Coconut, Cranberries, Hazel Nuts, Peanuts
1  3 Chocolate, Raisins <vs> Apple, Brown Sugar, Peanuts, Walnuts
1  4 Chocolate, Coconut <vs> Almonds, Apple, Brown Sugar, Cranberries
1  5 Hazel Nuts, Peanuts <vs> Almonds, Apple, Banana Chips, Raisins
1  6 Banana Chips, Coconut, Raisins <vs> Almonds, Cashews, Chocolate
1  7 Brown Sugar, Cashews, Hazel Nuts, Peanuts, Pecans <vs> Almonds
1  8 Cranberries, Raisins <vs> Apple, Chocolate, Hazel Nuts, Pecans
1  9 Pecans, Walnuts <vs> Banana Chips, Brown Sugar, Coconut, Raisins
1 10 Chocolate, Coconut, Hazel Nuts <vs> Almonds, Pecans, Walnuts
1 11 Cashews <vs> Brown Sugar, Cranberries, Hazel Nuts, Raisins, Walnuts
1 12 Peanuts, Walnuts <vs> Apple, Cashews, Coconut, Pecans
1 13 Apple, Cranberries, Walnuts <vs> Almonds, Banana Chips, Cashews
```

Obviously, this is a crude portrayal of the choice sets, but all the information is there. It is important to look to see if this design looks reasonable for your purposes. The presentation of the design to the subjects is much more involved than this, but the essential elements are here. Subjects are presented with two alternatives and asked to pick one of the two.

The data are collected and read into a SAS data set as follows:

```
   data chdata;
      input Block Sub (c1-c13) (1.) @@;
      datalines;
    1  1 1212221112122 1  2 2222221112122 1  3 2221222112122 1  4 2211221112121
    1  5 2212222112212 1  6 2212222122122 1  7 2222221112122 1  8 2122222112122
    1  9 2122221112122 1 10 2212221112122 1 11 1222222112122 1 12 2221222112122
    1 13 1212221112212 1 14 2112222122122 1 15 2212221112222 1 16 2212221112122
    1 17 2212221112122 1 18 2212221112122 1 19 2222221112222 1 20 2212221112122
    2  1 1212112212122 2  2 1112222212122 2  3 1212222112122 2  4 1112222212122
    2  5 1221112122122 2  6 1112212122122 2  7 1212112212122 2  8 1212212212122
    2  9 1112222222122 2 10 1211122212122 2 11 1112222222122 2 12 1212112222112
    2 13 1112222212122 2 14 1212122222122 2 15 1222222222122 2 16 1212122222112
    2 17 1212212212122 2 18 1112122212122 2 19 1112222222122 2 20 1112122212122
    3  1 1221211112222 3  2 1222221111222 3  3 2121211112222 3  4 1221211112222
    3  5 1222212122222 3  6 1122211121222 3  7 1222211112222 3  8 1121211112222
    3  9 1222211112222 3 10 1221211111222 3 11 2121211111222 3 12 1121221112222
    3 13 1122211112222 3 14 1122211122222 3 15 1121212111222 3 16 1121211122222
    3 17 1122211112222 3 18 1122211112222 3 19 1122211122222 3 20 1112211121222
    4  1 2211212112212 4  2 2111212122212 4  3 2111212222212 4  4 2111211122212
    4  5 2111222122212 4  6 2122212112212 4  7 2221212122221 4  8 2111212121222
    4  9 2121212111212 4 10 2122212121211 4 11 2121212111211 4 12 2111212122212
    4 13 2111212111211 4 14 2221212122212 4 15 2211211122212 4 16 2111212111212
    4 17 2211212121221 4 18 2121212112212 4 19 2112212111212 4 20 2111212121221
    5  1 2112122112112 5  2 1212122122211 5  3 2222122111111 5  4 2212222121211
    5  5 2222122112111 5  6 2122122212112 5  7 1212122111112 5  8 2222222122111
    5  9 1222222112111 5 10 2212122111111 5 11 2211221122111 5 12 2211121111111
    5 13 2212222221212 5 14 2222222112111 5 15 2222222121111 5 16 2222222111111
    5 17 2222122122211 5 18 1212122112111 5 19 2212222122111 5 20 2212222212111
    6  1 1222221212112 6  2 1222221212122 6  3 1222121212212 6  4 1222221212212
    6  5 1222221211112 6  6 1222221212112 6  7 1222221112122 6  8 1222221212112
    6  9 1221221211112 6 10 1222121222112 6 11 1222221211122 6 12 1221221212212
    6 13 1222121222212 6 14 1222221211112 6 15 1221121212112 6 16 1222221211212
    6 17 1222121212212 6 18 1221221212112 6 19 1212121212212 6 20 1222221212112
    7  1 2112222221211 7  2 1122222222111 7  3 1112222222111 7  4 2112222222111
    7  5 2112222222112 7  6 2112222222111 7  7 2112222222111 7  8 1112222221211
    7  9 2112222122111 7 10 1122222222211 7 11 2112222222211 7 12 2112222222121
    7 13 2112222222211 7 14 2112222222211 7 15 1112222222111 7 16 1112122222111
    7 17 2112222222111 7 18 2112222222211 7 19 2112222212211 7 20 1112222222211
    8  1 2211112221122 8  2 2212212222121 8  3 2212212221121 8  4 2212212121122
    8  5 2211122221121 8  6 2212222121121 8  7 2221122221111 8  8 2211212221122
    8  9 2211212221121 8 10 2211112221111 8 11 2211212221121 8 12 2212222221121
    8 13 2211112221122 8 14 2111222221122 8 15 2212222221121 8 16 2212222221121
    8 17 2222222221121 8 18 2111112121122 8 19 2111212221122 8 20 2112212222121
   ;
```

The data consist of a block number, a subject number, and then 13 choices, one for each of the 13 sets within each block. In the interest of space, data from four subjects appear on a single line. The following steps merge the data and the design and do the analysis:

```
%mktmerge(design=sasuser.chdes,     /* input final blocked choice design    */
          data=chdata,              /* input choice data                    */
          out=desdata,              /* output design and data               */
          blocks=block,             /* the blocking variable is block       */
          nsets=13,                 /* 13 choice sets per subject           */
          nalts=2,                  /* 2 alternatives in each set           */
          setvars=c1-c13)           /* the choices for each subject vars    */

%phchoice(on)                       /* customize PHREG for a choice model   */

proc phreg brief data=desdata;      /* provide brief summary of strata      */
   ods output parameterestimates=pe;/* output parameter estimates           */
   class x1-x13 / ref=first;        /* name all as class vars, '1' ref level*/
   model c*c(2) = x1-x13;           /* 1 - chosen, 2 - not chosen           */
                                    /* x1-x13 are independent vars          */
   label x1  = 'Almonds'            /* set of descriptive labels            */
         x2  = 'Apple'
         x3  = 'Banana Chips'
         x4  = 'Brown Sugar'
         x5  = 'Cashews'
         x6  = 'Chocolate'
         x7  = 'Coconut'
         x8  = 'Cranberries'
         x9  = 'Hazel Nuts'
         x10 = 'Peanuts'
         x11 = 'Pecans'
         x12 = 'Raisins'
         x13 = 'Walnuts';
   strata block sub set;            /* set within subject within block      */
   run;                             /* identify each choice set             */

proc sort data=pe;                  /* process the parameter estimates      */
   by descending estimate;          /* table by sorting by estimate         */
   run;

data pe;                            /* also get rid of the '2' level        */
   set pe;                          /* in the label                         */
   substr(label, length(label)) = ' ';
   run;

proc print label;                   /* print estimates with largest first   */
   id label;
   label label = '00'x;
   var df -- probchisq;
   run;

%phchoice(off)                      /* restore PHREG to a survival PROC     */
```

The results are as follows:

```
                              Cereal Bars

                          The PHREG Procedure

                           Model Information

                 Data Set                    WORK.DESDATA
                 Dependent Variable          c
                 Censoring Variable          c
                 Censoring Value(s)          2
                 Ties Handling               BRESLOW

                Number of Observations Read        4160
                Number of Observations Used        4160

                        Class Level Information

                                          Design
                      Class     Value     Variables

                      x1        1                 0
                                2                 1

                      x2        1                 0
                                2                 1

                      x3        1                 0
                                2                 1

                      x4        1                 0
                                2                 1

                      x5        1                 0
                                2                 1

                      x6        1                 0
                                2                 1

                      x7        1                 0
                                2                 1

                      x8        1                 0
                                2                 1

                      x9        1                 0
                                2                 1

                      x10       1                 0
                                2                 1
```

```
                         x11       1                 0
                                   2                 1

                         x12       1                 0
                                   2                 1

                         x13       1                 0
                                   2                 1

                                        Cereal Bars

                                    The PHREG Procedure

              Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                        Number of        Number of           Chosen          Not
             Pattern      Choices     Alternatives     Alternatives       Chosen

                   1         2080                2                1            1

                                     Convergence Status

                      Convergence criterion (GCONV=1E-8) satisfied.

                                   Model Fit Statistics

                                             Without           With
                           Criterion      Covariates     Covariates

                           -2 LOG L         2883.492       1414.992
                           AIC              2883.492       1440.992
                           SBC              2883.492       1514.314

                           Testing Global Null Hypothesis: BETA=0

                   Test                 Chi-Square       DF     Pr > ChiSq

                   Likelihood Ratio      1468.4999       13         <.0001
                   Score                 1083.7833       13         <.0001
                   Wald                   551.9476       13         <.0001
```

```
                        Type 3 Tests

                                   Wald
                Effect      DF    Chi-Square    Pr > ChiSq

                x1           1      201.2893        <.0001
                x2           1        3.2314        0.0722
                x3           1       12.5575        0.0004
                x4           1       65.4783        <.0001
                x5           1      394.3608        <.0001
                x6           1        0.9942        0.3187
                x7           1       17.9353        <.0001
                x8           1      183.3954        <.0001
                x9           1      132.2685        <.0001
                x10          1       20.2881        <.0001
                x11          1       46.2902        <.0001
                x12          1      123.3861        <.0001
                x13          1       92.4112        <.0001
                        Cereal Bars

                    The PHREG Procedure

             Multinomial Logit Parameter Estimates

                       Parameter      Standard
                 DF     Estimate         Error    Chi-Square    Pr > ChiSq

Almonds 2         1      1.67575       0.11811      201.2893        <.0001
Apple 2           1     -0.18096       0.10067        3.2314        0.0722
Banana Chips 2    1      0.36271       0.10235       12.5575        0.0004
Brown Sugar 2     1     -0.79392       0.09811       65.4783        <.0001
Cashews 2         1      2.95195       0.14865      394.3608        <.0001
Chocolate 2       1     -0.10184       0.10213        0.9942        0.3187
Coconut 2         1     -0.41696       0.09845       17.9353        <.0001
Cranberries 2     1      1.56238       0.11537      183.3954        <.0001
Hazel Nuts 2      1     -1.23998       0.10782      132.2685        <.0001
Peanuts 2         1      0.46061       0.10226       20.2881        <.0001
Pecans 2          1      0.72250       0.10619       46.2902        <.0001
Raisins 2         1      1.13299       0.10200      123.3861        <.0001
Walnuts 2         1      1.01308       0.10539       92.4112        <.0001
```

```
                                Cereal Bars

                             Parameter      Standard                     Pr >
                      DF      Estimate         Error    Chi-Square      ChiSq

        Cashews        1       2.95195       0.14865      394.3608     <.0001
        Almonds        1       1.67575       0.11811      201.2893     <.0001
        Cranberries    1       1.56238       0.11537      183.3954     <.0001
        Raisins        1       1.13299       0.10200      123.3861     <.0001
        Walnuts        1       1.01308       0.10539       92.4112     <.0001
        Pecans         1       0.72250       0.10619       46.2902     <.0001
        Peanuts        1       0.46061       0.10226       20.2881     <.0001
        Banana Chips   1       0.36271       0.10235       12.5575     0.0004
        Chocolate      1      -0.10184       0.10213        0.9942     0.3187
        Apple          1      -0.18096       0.10067        3.2314     0.0722
        Coconut        1      -0.41696       0.09845       17.9353     <.0001
        Brown Sugar    1      -0.79392       0.09811       65.4783     <.0001
        Hazel Nuts     1      -1.23998       0.10782      132.2685     <.0001
```

The number of observations read and used is 4160. This is 20 subjects times 8 blocks times 13 sets per block, times 2 alternatives. Also, the data consist of 2080 (4160 divided by 2 alternatives) choice sets where two alternatives were presented, one was chosen, and one was not chosen. The parameter estimate table is displayed twice. Once in the original order, the order of the attributes, and once sorted by the parameter estimates. Cashews are most preferred, and Hazel Nuts are least preferred.

Example 8, A MaxDiff Choice Experiment

This example is like the previous example in that we will use the same ingredients of high-end cereal bars as before. These ingredients are as follows:

Almonds
Apple
Banana Chips
Brown Sugar
Cashews
Chocolate
Coconut
Cranberries
Hazel Nuts
Peanuts
Pecans
Raisins
Walnuts

This time, we will show subjects sets of attributes (or ingredients) and ask them to pick the one they like the best and the one they like the least. This is called a MaxDiff or best-worst study (Louviere 1991, Finn and Louviere 1992). We will use a balanced incomplete block design. The $t = 13$ attributes are shown in b sets of size k.

You can find the sizes in which a BIBD might be available for ranges of t, b, and k, using the `%MktBSize` macro as in the following example:

```
title 'Cereal Bars';

%mktbsize(nattrs=13,                /* 13 total attributes                 */
          setsize=2 to 6,           /* show between 2 and 6 at once        */
          nsets=2 to 40,            /* make between 2 and 40 choice sets   */
          options=ubd,              /* consider unbalanced designs         */
          maxreps=5)                /* permit multiple replications, which */
                                    /* will show us some BIBDs that might  */
                                    /* not be otherwise listed with        */
                                    /* options=ubd                         */
```

The results of this step are as follows:

```
                                   Cereal Bars

                                                          n
 t            k     b         r           Lambda          Total
 Number of    Set   Number    Attribute   Pairwise        Sample    Number of
 Attributes   Size  of Sets   Frequency   Frequencies     Size      Replications

     13         2      13          2          0.17           26          1
     13         2      26          4          0.33           52          2
     13         2      39          6           0.5           78          3
     13         3      13          3           0.5           39          1
     13         3      26          6             1           78          2
     13         3      39          9           1.5          117          3
     13         4      13          4             1           52          1
     13         4      26          8             2          104          2
     13         4      39         12             3          156          3
     13         5      13          5          1.67           65          1
     13         5      26         10          3.33          130          2
     13         5      39         15             5          195          3
     13         6      13          6           2.5           78          1
     13         6      26         12             5          156          2
     13         6      39         18           7.5          234          3
```

A BIBD might be possible with all four values of k, the set size shown. We could do a small pilot study with 13 choice sets and 4 attributes shown. The following step constructs the BIBD:

```
%mktbibd(out=sasuser.bibd,          /* output BIBD                         */
         nattrs=13,                 /* 13 total attributes                 */
         setsize=4,                 /* show 4 in each set                  */
         nsets=13,                  /* 13 choice sets                      */
         seed=93)                   /* random number seed                  */
```

The results are as follows:

```
                              Cereal Bars

            Block Design Efficiency Criterion       100.0000
            Number of Attributes, t                       13
            Set Size, k                                    4
            Number of Sets, b                             13
            Attribute Frequency                            4
            Pairwise Frequency                             1
            Total Sample Size                             52
            Positional Frequencies Optimized?            Yes

                   Attribute by Attribute Frequencies

                   1  2  3  4  5  6  7  8  9 10 11 12 13

                1  4  1  1  1  1  1  1  1  1  1  1  1  1
                2     4  1  1  1  1  1  1  1  1  1  1  1
                3        4  1  1  1  1  1  1  1  1  1  1
                4           4  1  1  1  1  1  1  1  1  1
                5              4  1  1  1  1  1  1  1  1
                6                 4  1  1  1  1  1  1  1
                7                    4  1  1  1  1  1  1
                8                       4  1  1  1  1  1
                9                          4  1  1  1  1
               10                             4  1  1  1
               11                                4  1  1
               12                                   4  1
               13                                      4

                   Attribute by Position Frequencies

                                 1  2  3  4

                              1  1  1  1  1
                              2  1  1  1  1
                              3  1  1  1  1
                              4  1  1  1  1
                              5  1  1  1  1
                              6  1  1  1  1
                              7  1  1  1  1
                              8  1  1  1  1
                              9  1  1  1  1
                             10  1  1  1  1
                             11  1  1  1  1
                             12  1  1  1  1
                             13  1  1  1  1
```

```
                    Cereal Bars
          Balanced Incomplete Block Design

                 x1    x2    x3    x4

                  3    11     1    10
                 11     4     7     6
                  4     9    13     3
                  8    10     6     9
                  7    13    10    12
                  5     1     9     7
                 10     2     5     4
                 13     5     8    11
                  1     6     2    13
                 12     8     4     1
                  2     7     3     8
                  9    12    11     2
                  6     3    12     5
```

A BIBD was found, and every attribute is shown with every other attribute exactly once. Furthermore, the positional frequencies are perfect. It is important to vary the positions in which the attributes are displayed.

The following step displays the BIBD, but with the actual attribute names rather than numbers:

```
data _null_;
   array alts[13] $ 12 _temporary_ ('Almonds' 'Apple' 'Banana Chips'
                  'Brown Sugar' 'Cashews' 'Chocolate' 'Coconut' 'Cranberries'
                  'Hazel Nuts' 'Peanuts' 'Pecans' 'Raisins' 'Walnuts');
   set sasuser.bibd;                /* read design                         */
   put alts[x1] +(-1) ', '          /* print each attr, comma separated    */
       alts[x2] +(-1) ', ' alts[x3] +(-1) ', ' alts[x4];
   run;
```

The results are as follows:

```
    Banana Chips, Pecans, Almonds, Peanuts
    Pecans, Brown Sugar, Coconut, Chocolate
    Brown Sugar, Hazel Nuts, Walnuts, Banana Chips
    Cranberries, Peanuts, Chocolate, Hazel Nuts
    Coconut, Walnuts, Peanuts, Raisins
    Cashews, Almonds, Hazel Nuts, Coconut
    Peanuts, Apple, Cashews, Brown Sugar
    Walnuts, Cashews, Cranberries, Pecans
    Almonds, Chocolate, Apple, Walnuts
    Raisins, Cranberries, Brown Sugar, Almonds
    Apple, Coconut, Banana Chips, Cranberries
    Hazel Nuts, Raisins, Pecans, Apple
    Chocolate, Banana Chips, Raisins, Cashews
```

Obviously, this is a crude portrayal of the choice sets, but all the information is there. It is important to look to see if this design looks reasonable for your purposes. The presentation of the design to the subjects is much more involved than this, but the essential elements are here. Subjects are presented with four ingredients and asked to pick the one they like the best and the one they like the least.

The data are entered into a SAS data set as follows:

```
data bwdata;
   input (x1-x26) (1.);
   datalines;
34133132412334244323323143
34143214411431314314422141
23143414421432344221422142
31123213421332231241432143
34133214421434242124233142
34314232211334211423432141
24434114312334344243122141
31134124322334214243213442
41123114131432241321422432
41423113341334431242424132
32423412122334244343423141
31433214431334231313122443
41133214411334244314422141
34213123431334241223212443
13134214311334211343422141
32242421341331232323122142
41124214311342244313132142
31143214411334231213411442
41423124211234244323412141
31233413211431121243424141
;
```

There are 20 rows for 20 subjects and 26 variables (13 best choices and 13 worst choices). The data alternate best then worst: `x1` is a best choice, `x2` is a worst choice, `x3` is a best choice, `x4` is a worst choice, and so on. The data all consist of integers in the range 1 to 4. These represent the positions of the chosen alternatives. There are many other ways the data could be handled. Best can come first or worst can come first, the variables can alternate or not, and the data could be positions (1-4 in this case) or the data could be attribute numbers (1-13 in this case). See the `%MktMDiff` macro on page 1105 for more information.

The data are analyzed as follows:

```
%let attrlist=Almonds,Apple,Banana Chips,Brown Sugar,Cashews,Chocolate
,Coconut,Cranberries,Hazel Nuts,Peanuts,Pecans,Raisins,Walnuts;

%phchoice(on)                     /* customize PHREG for a choice model   */

%mktmdiff(bwaltpos,               /* data are best then worst and         */
                                  /* alternating and are the positions    */
                                  /* of the chosen attributes             */
          nattrs=13,              /* 13 attributes                        */
          nsets=13,               /* 13 choice sets                       */
          setsize=4,              /* 4 attributes shown in each set       */
          attrs=attrlist,         /* list of attribute names              */
          data=bwdata,            /* input data set with data             */
          design=sasuser.bibd)    /* input data set with BIBD             */
```

First, the `%PHChoice` macro is used to customize the output from PROC PHREG, which is called by the `%MktMDiff` macro, for the multinomial logit model. Next, the description of each attribute is stored in a macro variable with commas delimiting the individual labels. Next, the `%MktMDiff` macro is called to combine the data and the design and do the analysis. The results are as follows:

```
                                  Cereal Bars
Var Order:   Best then Worst
Alternating: Variables Alternate
Data:        Positions (Not Attribute Numbers)
Best Vars:   x1 x3 x5 x7 x9 x11 x13 x15 x17 x19 x21 x23 x25
Worst Vars:  x2 x4 x6 x8 x10 x12 x14 x16 x18 x20 x22 x24 x26
Attributes:  Almonds
             Apple
             Banana Chips
             Brown Sugar
             Cashews
             Chocolate
             Coconut
             Cranberries
             Hazel Nuts
             Peanuts
             Pecans
             Raisins
             Walnuts
```

```
                              Cereal Bars

                          The PHREG Procedure

                           Model Information

                    Data Set                 WORK.CODED
                    Dependent Variable       c
                    Censoring Variable       c
                    Censoring Value(s)       2
                    Frequency Variable       Count
                    Ties Handling            BRESLOW

                  Number of Observations Read         191
                  Number of Observations Used         191
                  Sum of Frequencies Read            2080
                  Sum of Frequencies Used            2080

       Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                 Number of       Number of          Chosen          Not
      Pattern      Choices    Alternatives    Alternatives       Chosen

            1           26              80              20           60

                           Convergence Status

               Convergence criterion (GCONV=1E-8) satisfied.

                         Model Fit Statistics

                                  Without           With
                  Criterion     Covariates     Covariates

                  -2 LOG L        4557.308       4135.039
                  AIC             4557.308       4159.039
                  SBC             4557.308       4210.085

                  Testing Global Null Hypothesis: BETA=0

          Test                 Chi-Square       DF     Pr > ChiSq

          Likelihood Ratio       422.2690       12         <.0001
          Score                  368.1231       12         <.0001
          Wald                   268.2494       12         <.0001
```

```
                                Cereal Bars
                    Multinomial Logit Parameter Estimates

                              Parameter      Standard
                        DF     Estimate         Error    Chi-Square    Pr > ChiSq

     Cashews             1      2.37882       0.35539       44.8047        <.0001
     Cranberries         1      0.49982       0.29266        2.9167        0.0877
     Almonds             1      0.46203       0.27989        2.7250        0.0988
     Raisins             1      0.18013       0.28220        0.4074        0.5233
     Walnuts             0            0             .             .             .
     Pecans              1     -0.46634       0.27270        2.9245        0.0872
     Peanuts             1     -0.57648       0.29073        3.9319        0.0474
     Chocolate           1     -1.26584       0.28645       19.5276        <.0001
     Banana Chips        1     -1.32289       0.28004       22.3153        <.0001
     Apple               1     -1.34142       0.29063       21.3033        <.0001
     Coconut             1     -1.83155       0.28743       40.6033        <.0001
     Brown Sugar         1     -1.96331       0.29142       45.3891        <.0001
     Hazel Nuts          1     -2.55326       0.29966       72.5995        <.0001
```

The first table provides a summary of the data and the specifications. The data alternate best then worst and are positions not attribute numbers. The best variables are the odd numbered variables, and the worst variables are the even numbered variables. Finally the attributes are listed. There were 26 choices (13 best and 13 worst) and 20 times (20 subjects) an alternative was chosen and 60 times (3 not chosen by 20 subjects) alternatives were not chosen. The final table of parameter estimates is displayed ordered in descending order of preference. Cashews are most preferred and Hazel nuts are least preferred by these subjects.

The design is arrayed so that there is one classification variable with 13 levels. The 'Walnuts' level, being the last level alphabetically, is the reference level and has a coefficient of 0. If by chance it had been the most preferred level, then all of the other coefficients would have been negative. If it had been the least preferred level, then all of the other coefficients would have been positive. This is illustrated in the following step:

```
%mktmdiff(bwaltpos,                  /* data are best then worst and        */
                                     /* alternating and are the positions   */
                                     /* of the chosen attributes            */
          nattrs=13,                 /* 13 attributes                       */
          nsets=13,                  /* 13 choice sets                      */
          setsize=4,                 /* 4 attributes shown in each set      */
          attrs=attrlist,            /* list of attribute names             */
          classopts=zero='Hazel Nuts',/* set the reference level            */
          data=bwdata,               /* input data set with data            */
          design=sasuser.bibd)       /* input data set with BIBD            */

%phchoice(off)                       /* restore PHREG to a survival PROC    */
```

Normally, you should not specify the `classopts=` option unless you are changing the reference level.

The last table of results is as follows:

```
                                   Cereal Bars

                        Multinomial Logit Parameter Estimates

                             Parameter      Standard
                     DF       Estimate         Error    Chi-Square    Pr > ChiSq

Cashews               1        4.93207       0.38051      168.0095        <.0001
Cranberries           1        3.05308       0.30862       97.8642        <.0001
Almonds               1        3.01529       0.31258       93.0567        <.0001
Raisins               1        2.73338       0.30734       79.0976        <.0001
Walnuts               1        2.55326       0.29966       72.5995        <.0001
Pecans                1        2.08691       0.29407       50.3637        <.0001
Peanuts               1        1.97678       0.30362       42.3879        <.0001
Chocolate             1        1.28741       0.28701       20.1202        <.0001
Banana Chips          1        1.23036       0.29306       17.6254        <.0001
Apple                 1        1.21184       0.28834       17.6636        <.0001
Coconut               1        0.72170       0.27811        6.7342        0.0095
Brown Sugar           1        0.58994       0.28726        4.2175        0.0400
```

Now, all but the reference level is shown, and all of the estimates are positive. The difference is they have all been shifted by subtracting the original coefficient for 'Hazel Nuts'. This makes the new coefficient for 'Hazel Nuts' zero (larger by the absolute value of the original coefficient) and the rest larger by the same amount.

Conclusions

This chapter introduced some choice design terminology and ideas with some examples but without going into great detail on how to make designs and process data for analysis. The information in this chapter should provide a good foundation for all of the detailed examples in the discrete choice chapter.

Choice Design Glossary

Experimental design, choice modeling, and choice design, like all other areas, all have their own vocabularies. This section defines some of those terms. These terms are used and defined throughout this chapter and the discrete choice chapter (pages 285–663).

aliased – Two effects are confounded or aliased when they are not distinguishable from each other. Lower-order effects such as main effects or two-way interactions might be aliased with higher-order interactions in most of our designs. We estimate lower-order effects by assuming that higher-order effects are zero or negligible. See page 495.

allocation study – An allocation study is a choice study where multiple, not single choices are made. For example, in prescription drug marketing, physicians are asked questions like "For the next ten prescriptions you write for a particular condition, how many would you write for each of these drugs?" See page 535.

alternative – An alternative is one of the options available to be chosen in a choice set. An alternative might correspond to a particular brand in a branded study or just a bundle of attributes in a generic study. See page 55.

alternative-specific attribute – An alternative-specific attribute is one that is expected to interact with brand. If you expect utility to change in different ways for the different brands, then the attribute is alternative-specific. Otherwise, it is generic. In the analysis, there is a set of alternative-specific attribute parameters for each alternative. See page 55.

asymmetric design – An experimental design where not all factors have the same number of levels. At least one factor has a number of levels that is different from at least one other factor. See page 112.

attribute – An attribute is one of the characteristics of an alternative. Common attributes include price, size, and a variety of other product-specific factors. See page 55.

availability cross-effects – A design might have a varying number of alternatives. When not all alternatives are available in every choice set, availability cross-effects, might be of interest. These capture the effects of the presence/absence of one brand on the utility of another. See page 470.

balance – A design is balanced when each level occurs equally often within each factor. See page 58.

balanced incomplete block design – A balanced incomplete block design (BIBD) is a list of t treatments that appear together in b blocks. Each block contains a subset ($k < t$) of the treatments. A BIBD is commonly represented by a $b \times k$ matrix with entries ranging from 1 to t. Each of the b rows is one block. Each of the t treatments must appear the same number of times in the design, and each of the t treatments must appear with each of the other $t - 1$ treatments in exactly the same number of blocks. When the treatment frequencies are constant in a block design, but the pairwise frequencies are not constant, the design is called an *unbalanced block design*. More generally, an *incomplete block design* includes BIBDs, unbalanced block designs, and block designs where neither the treatment nor the pairwise frequencies are constant. BIBDs and unbalanced block designs are used in marketing research for MaxDiff studies (see the `%MktMDiff` macro, page 1105) and partial-profile designs (see the `%MktPPro` macro, page 1145). In a MaxDiff study, there are t attributes shown in b sets of size k. In a certain class of partial-profile designs, there are t attributes, shown in b blocks of choice sets, where k attributes vary in each block. Block designs can be constructed with the `%MktBIBD` macro (see page 963). Also see page 989 for information about when a BIBD might exist. See page 115.

binary coding – Binary coding replaces the levels of qualitative or `class` variables with binary indicator variables. Less-than-full-rank binary coding creates one binary variable for each level of the factor. Full-rank binary coding (or reference cell coding) creates one binary variable for all but one level, the reference level. See page 73.

blocking – Large choice designs need to be broken into blocks. Subjects will just see a subset of the full design. How many blocks depends on the number of choice sets and the complexity of the choice task. See pages 217 and 426.

branded design – A branded choice design has one factor that consists of a brand name or other alternative label. The vacation examples on pages 339-443 are examples of branded designs even though the labels, destinations, and not brands. The examples starting on pages 302, 468, and 444 use branded designs and actual brand names.

canonical correlation – The first canonical correlation is the maximum correlation that can occur between linear combinations of two sets of variables. We use the canonical correlation between two sets of coded `class` variables as a way of showing deviations from design orthogonality. See page 101.

choice design – A choice design has one column for every different product attribute and one row for every alternative of every choice set. In some cases, different alternatives have different attributes and different choice sets might have differing numbers of alternatives. See pages 55 and 67–71.

choice set – A choice set consists of two or more alternatives. Subjects see one or more choice sets and choose one alternative from each set. See page 55.

confounded – See "aliased."

covariance matrix – See "variance matrix."

cross-effects – A cross-effect represents the effect of one alternative on the utility of another alternative. When the IIA assumption holds, all cross-effects are zero. See page 452.

deviations from means coding – See "effects coding."

efficiency – Efficiency is a scale or measurement of the goodness of an experimental design based on the average of the eigenvalues of the variance matrix. *A*-efficiency is a function of the arithmetic mean of the eigenvalues, which is also the arithmetic mean of the variances. *D*-efficiency is a function of the geometric mean of the eigenvalues. In many cases, efficiency is scaled to a 0 to 100 scale where 0 means one or more parameters cannot be estimated and 100 means the design is perfect. See page 62.

effects coding – Effects coding (or deviations from means coding) is similar to full-rank binary coding, except that the row for the reference level is set to all –1's instead of all zeros. See page 73.

experimental design – An experimental design is a plan for running an experiment. See page 53.

factor – A factor is a column of an experimental design with two or more fixed values, or levels. In the context of conjoint and choice modeling, you could use the terms "factor" and "attribute" interchangeably. However, in this book, the term "factor" is usually used to refer to a column of a "raw" design (with columns such as `x1` and `x2`) that has not yet been processed and relabeled into the form of a conjoint or choice design. It is also used when discussing coding and other design concepts that are the same for linear model, choice, and conjoint designs. See page 54.

fractional-factorial design – A fractional-factorial design is a subset of a full-factorial design. Often, this term is used to refer to particularly "nice" fractions such as the designs created by PROC FACTEX. See page 995.

full-factorial design – A full-factorial design consists of all possible combinations of the all of the levels of all of the factors. See page 57.

generic attribute – A generic attribute is one that is not expected to interact with brand (or more generally, the attribute label). If you expect utility to change as a function of the levels of the attribute in the same way for every brand, then the attribute is generic. In contrast, if you expect utility to change in different ways for the different brands, then the attribute is alternative-specific. All attributes in generic designs are generic. In the analysis, there is one set of parameters for generic attributes, regardless of the number of alternatives. See page 55.

generic design or **generic model** – A generic design has no brands or labels for the alternatives. The alternatives are simply bundles of attributes. For example, each alternative might be a cell phone or computer all made by the same manufacturer. See page 102.

IIA – The independence of irrelevant alternatives or IIA property states that utility only depends on an alternative's own attributes. IIA means the odds of choosing alternative c_i over c_j do not depend on the other alternatives in the choice set. Departures from IIA exist when certain subsets of brands are in more direct competition and tend to draw a disproportionate amount of share from each other than from other brands. See pages 452, 459, 468, 674, and 679.

incomplete block design – A block design where treatment frequencies and pairwise frequencies are not constant, or an unbalanced block design, or a balanced incomplete block design. Usually, the term "incomplete block design" is used to refer to block designs that do not meet the stricter criteria necessary to be classified as an unbalanced block design or a balanced incomplete block design. See "balanced incomplete block design" for more information.

indicator variables – Indicator variables (or "dummy variables") are binary variables that are used to represent categorical or `class` variables in an analysis. Less precisely, this term is sometimes used to refer to other coding schemes. See page 73.

information matrix – The information matrix (for factorial design matrix $\mathbf{X}$) is $\mathbf{X}'\mathbf{X}$. See page 62.

interaction – Interactions involve two or more factors, such as a brand by price interaction. For example, in a model with interactions brand preference is different at the different prices and the price effect is different for the different brands. See page 57.

level – A level is a fixed value of a design factor. Raw designs typically start with levels that are positive or nonnegative integers. Then these levels are reassigned with actual levels such as brands or prices. See page 53.

linear arrangement – The linear arrangement of a choice design ("linear arrangement" for short) contains one row for each choice set and one column for every attribute of every alternative. However, brand or some other alternative-labeling factor, is not a factor in the linear arrangement. Rather, the brand or alternative label is a bin into which the other factors are collected. The columns are grouped, the first group contains every attribute for the first alternative, ..., and the *jth* group contains every attribute for the *jth* alternative. The linear arrangement is an intermediate step in constructing a choice design by one of the several available approaches. In the linear arrangement, all of the information for a single choice set is arrayed in a single line or row vector. You can rearrange the design from the linear arrangement to the standard choice design arrangement by moving each of the m blocks for the m alternatives below the preceding block creating a choice design with m times as many rows as previously and approximately $1/m$ times as many columns. In other words, in the linear arrangement, there is one row per choice set, and in the choice design arrangement there is one matrix with m rows per choice set. See pages 67–71.

linear design – A term used in previous editions for what is now called the linear arrangement of a choice design or "linear arrangement" for short.

main effect – A main effect is a simple effect, such as a price or brand effect. For example, in a main-effects model the brand effect is the same at the different prices and the price effect is the same for the different brands. See page 57.

MaxDiff – In a MaxDiff study, subjects are shown sets of messages or product attributes and are asked to choose the best (or most important) from each set as well as the worst (or least important). A balanced incomplete block design is used, and the data are analyzed with a choice model. See page 225.

mother logit model – The mother logit model is a model with cross-effects that can be used to test for violations of IIA. See page 452.

orthogonal – When every pair of levels occurs equally often across all pairs of factors, the design is orthogonal. Another way in which a design can be orthogonal is when the frequencies for level pairs are proportional instead of equal. See page 58.

orthogonal array – An orthogonal array is an experimental design in which all estimable effects are uncorrelated. See page 59.

orthogonal contrast coding – Orthogonal contrast coding codes all but the reference level as an orthogonal contrast between each level and the levels that come before along with the reference level. The coded values are all integers. See page 73. Also see "standardized orthogonal contrast coding."

partial-profile design – A partial-profile choice design consists of bundles of attributes where only a subset of attributes vary in each choice set. Partial-profile designs can be constructed from an orthogonal array and a balanced incomplete block design. See page 207. Alternatively, they can be constructed by creating designs with restrictions. See page 595.

random number seed – The random number seed is an integer in the range 1 to 2,147,483,646 that is used to provide a starting point for the random number stream. While our designs are not random, there is often some random process used in their creation. See page 94.

randomization – Randomization involves sorting the rows of a design into a random order and randomly reassigning all of the factor levels. See page 57.

reference cell coding – See "binary coding."

reference level – The reference level is the level of a factor that does not correspond to a binary variable in reference level (binary, or indicator variable) coding. In effects or the orthogonal codings, it is the level that corresponds to the row of –1's. The reference level is by default the last level of the factor, but you can change that with the `zero=` option. See page 73.

resolution – Resolution identifies which effects are estimable. For resolution III designs, all main effects are estimable free of each other, but some of them are confounded with two-factor interactions. For resolution IV designs, all main effects are estimable free of each other and free of all two-factor interactions, but some two-factor interactions are confounded with other two-factor interactions. For resolution V designs, all main effects and two-factor interactions are estimable free of each other. See page 58.

runs – A run is a row of an experimental design. See page 53.

seed – See "random number seed."

standardized orthogonal contrast coding – Standardized orthogonal contrast coding codes all but the reference level as an orthogonal contrast between each level and the levels that come before along with the reference level. The coded values are scaled so that the sum of squares of each column equals the number of levels. See page 73. Also see "orthogonal contrast coding."

symmetric design – An experimental design where all factors have the same number of levels. See page 112.

unbalanced block design – An incomplete block design where every treatment appears with the same frequency, but pairwise frequencies are not constant. See "balanced incomplete block design" for more information.

variance matrix – The variance matrix (for linear model design matrix $\mathbf{X}$) is proportional to $(\mathbf{X}'\mathbf{X})^{-1}$. See page 62. Also see page 71 to see the variance matrix for a choice model.

Exercises

These exercises are designed to acquaint you with the basic principles of designing a choice experiment. You might find it helpful to store each choice design in a permanent SAS data set, each with a unique name, so that you can compare the designs when you are done. You should also specify random number seeds throughout so that you can reproduce your results.

Here are some questions to ask yourself throughout. What coding should I use? Does it matter? Why? What is the range of D-efficiency with this coding? Is this a good candidate set size? Might I do better with a smaller candidate set? Might I do better with a larger candidate set? Might I do better with a larger design and more blocks? Might I do better with a smaller design? Should I use the randomized design? Does it matter?

Do not expect the answers to these or some of the questions below (such as some of the "Why?" questions) to always be straight-forward. There might not be a clear correct answer. Sometimes you have to make judgment calls.

If any step takes more than a minute or two of computer time, you should go back and try to simplify your approach. Typically, you should start with a very small candidate set and a small number of iterations. Make your first design for each problem as quickly as possible, even if it is not optimal. You can go back later and try more iterations or larger candidate sets.

Direct Construction of a Generic Design

1) In this exercise, you will make a generic choice design with 6 alternatives, 6 choice sets, and six-level attributes. Assume a main-effects model with $\boldsymbol{\beta} = \mathbf{0}$.

1.a) What is the maximum number of six-level attributes that you can create in a choice design with perfect 100% relative D-efficiency?

1.b) Make and display an optimal generic design with the maximum number of attributes determined in the previous question. How often does each level of each attribute appear with each level of each other attribute across all alternatives and choice sets?

1.c) Evaluate the D-efficiency of the choice design.

1.d) What are the variances? What is D-error? What is D-efficiency? How are these quantities related?

1.e) How many parameters are in the model? Why? What is the maximum number of parameters you could estimate with 6 alternatives and 6 choice sets? Why?

1.f) Make a generic choice design with 6 alternatives, 6 choice sets, and 6 six-level attributes using the same approach that you used to make the previous design. That is, just add more attributes to the code you already have. Evaluate it and its D-efficiency.

1.g) How many parameters are in the model? Why? How many more parameters could you add to the model?

1.h) If a design existed for this specification with 100% relative D-efficiency, what would the raw unscaled D-efficiency be?

1.i) Would you use this design in a real study? Why or why not? What might you consider changing to make a better design?

Generic Design Construction by Searching Candidate Alternatives

2) In this exercise, you will again make a generic choice design with 6 alternatives, 6 choice sets, and 6 six-level attributes, but by using a different method. Assume a main-effects model with $\boldsymbol{\beta} = \mathbf{0}$.

2.a) Construct this design by searching a candidate set of alternatives. What is the *D*-efficiency?

2.b) Try at least two other candidate sets. What sizes did you pick? Why? Which one works best?

2.c) Construct the relative *D*-efficiency of each design (including the corresponding design from exercise 1 relative to each other using the following program:

```
proc iml;
   eff = { };   /* insert list of efficiencies inside of braces:
                   example: eff = { 5.2 3.1 4.3 3.5};
                 */
   label = { }; /* provide labels for each design showing the candidate
                   set sizes.  example:
                   label = {"exercise 1"   "cand = a"
                            "cand = b"     "cand = c"};
                 */
   x = j(ncol(eff), ncol(eff), 0);
   do i = 1 to ncol(eff);
      do j = 1 to ncol(eff);
         x[i,j] = 100 # eff[i] / eff[j];
         end;
      end;
   print x[rowname=label colname=label];
   quit;
```

2.d) Which design is best? Why?

Generic Design Construction by Searching Candidate Choice Sets

3) In this exercise, you will again make a generic choice design with 6 alternatives, 6 choice sets, and 6 six-level attributes, but again by using a different method. Assume a main-effects model with $\boldsymbol{\beta} = \mathbf{0}$.

3.a) Construct this design by searching a candidate set of choice sets. What is the *D*-efficiency? Hint: use the `%MktEx` options `options=quickr largedesign`, `maxtime=1`. See the macro documentation chapter to answer the next question: What do these options do, and why do we use them here?

3.b) Add the *D*-efficiency from this design to your PROC IML program and re-run the program. What are the results? What is the best approach for this problem?

Symmetric Alternative-Specific Designs

4) In this exercise, you will create a choice design for a study with four brands, A, B, C, and D. Each choice set will have four alternatives, and each of the four brands will always appear in each choice set. Each brand has 4 four-level attributes. You should begin by using the `%MktEx` and `%MktRoll` macros to make a linear arrangement and convert it into a choice design.

4.a) What is the minimum number of choice sets that you need? How many would you choose? Would you block the design? If so, how many blocks would you choose? What other block sizes might you consider?

4.b) Construct and display the linear and choice designs.

4.c) Evaluate the efficiency of the choice design. Assume a main-effects model with $\boldsymbol{\beta} = \mathbf{0}$. How many parameters are in the model? Why? What is the structure of the variance matrix?

4.d) Evaluate the efficiency of the choice design. Assume an alternative-specific effects model with $\boldsymbol{\beta} = \mathbf{0}$. How many parameters are in the model? Did you get 51? Why? What is the structure of the variance matrix?

4.e) What are the variances? What is *D*-efficiency? Is this design optimal? What are its strengths? What are its weaknesses?

4.f) Again, construct a choice experiment for four brands, each with 4 four-level attributes, for an alternative-specific model with $\boldsymbol{\beta} = \mathbf{0}$. However, this time search a candidate set of alternatives. What is the structure of the variance matrix?

4.g) Try several candidate set sizes. Compare your *D*-efficiencies from the linear arrangement and the candidate set search approach using the IML program from exercise 2. Which method works best? Why?

4.h) Which approach has the "nicest" variances and covariances? What is the range of relative *D*-efficiency in this problem?

4.i) Block the best design that you found.

4.j) Again, construct a choice experiment for four brands, each with 4 four-level attributes, for a main-effects model with $\boldsymbol{\beta} = \mathbf{0}$. Search a candidate set of alternatives, but this time create only three alternatives. Each choice set will have between 1 and 3 brands, and at least one brand will be missing from each choice set.

4.k) Does brand ever appear more than once in a choice set? Why or why not?

4.l) Once again, construct a choice experiment for four brands, each with 4 four-level attributes, for an alternative-specific model with $\boldsymbol{\beta} = \mathbf{0}$. Search a candidate set of alternatives. This time, the first attribute is price. Construct a design such that the prices are as follows: Brand A's prices are 1.49, 1.99, 2.49, and 2.99; Brand B's prices are 1.99, 2.49, 2.99, and 3.49; Brand C's prices are 1.79, 2.29, 2.79, and 3.29; and Brand D's prices are 1.69, 2.19, 2.69, and 3.19. Hints: Use a DATA step to convert the `x1` factor into a price attribute with different prices for each alternative. You can use the `drop=` option in the `%ChoicEff` macro to drop extra terms from the model. What is the difference between your designs with and without the extra terms dropped?

Asymmetric Alternative-Specific Designs

5) In this exercise, you will create a choice design for a study with four brands, A, B, C, and D. Each choice set will have four alternatives, and each of the four brands will always appear in each choice set. Each brand has a four-level attribute, 2 three-level attributes, and a two-level attributes. You will begin by using the `%MktEx` and `%MktRoll` macros to make a linear arrangement and convert it into a choice design.

5.a) What is the minimum number of choice sets that you need? How many would you choose? Would you block the design? If so, how many blocks would you choose? What other block sizes might you consider?

5.b) Construct and display the linear and choice designs.

5.c) Evaluate the efficiency of the choice design. Assume a main-effects model with $\boldsymbol{\beta} = \mathbf{0}$. How many parameters are in the model? Why?

5.d) Evaluate the efficiency of the choice design. Assume an alternative-specific effects model with $\boldsymbol{\beta} = \mathbf{0}$. How many parameters are in the model? Why?

5.e) What are the variances? What is *D*-efficiency? Is this design optimal? What are its strengths? What are its weaknesses?

5.f) Again, construct a choice experiment for four brands, each with 4 four-level attributes, for an alternative-specific model with $\boldsymbol{\beta} = \mathbf{0}$. However, this time search a candidate set of alternatives.

5.g) Try several candidate set sizes. Compare your *D*-efficiencies from the linear arrangement and the candidate set search approach using the IML program from exercise 2. Which method works best? Why? What is the range of relative *D*-efficiency in this problem?

Restricted Designs

6) Construct a choice experiment for three brands, each with 4-, 3-, 3-, and 2-level attributes, for a main-effects model with $\boldsymbol{\beta} = \mathbf{0}$. Search a candidate set of choice sets, each with three alternative, one per brand. Disallow from consideration any choice set where the four-level attribute has the same level in two or more alternatives.

6.a) How many candidate choice sets would you try first? What other choices are worth a try? What is the smallest number that you could try? What is the largest number that you would try?

6.b) Try several different choice set sizes. Which works best? Why? Did the restrictions get imposed correctly?

6.c) How many parameters are in the choice model? What is the maximum number for this specification? Why? Would you feel comfortable using the best design that you found?

6.d) Construct a choice experiment for the same three brands, again with 4-, 3-, 3-, and 2-level attributes, for a main-effects model with $\boldsymbol{\beta} = \mathbf{0}$. Search a candidate set of alternatives. Disallow from consideration any choice set where the three- or four-level attributes have the same level in two or more alternatives. Write a restrictions macro without `do` loops.

6.e) Repeat exercise 6.d, but this time use 3 `do` loops and a single assignment statement with one simple Boolean expression. Do you get the same results as 6.d?

Answers appear in `http://support.sas.com/techsup/technote/mr2010c.sas`.

Efficient Experimental Design with Marketing Research Applications

Warren F. Kuhfeld

Randall D. Tobias

Mark Garratt

Abstract

We suggest using *D*-efficient experimental designs for conjoint and discrete-choice studies, and discuss orthogonal arrays, nonorthogonal designs, relative efficiency, and nonorthogonal design algorithms. We construct designs for a choice study with asymmetry and interactions and for a conjoint study with blocks and aggregate interactions.[*]

Introduction

The design of experiments is a fundamental part of marketing research. Experimental designs are required in widely used techniques such as preference-based conjoint analysis and discrete-choice studies (e.g., Carmone and Green 1981; Elrod, Louviere, and Davey 1992; Green and Wind 1975; Huber, et al. 1993; Lazari and Anderson 1994; Louviere 1991; Louviere and Woodworth 1983; Wittink and Cattin 1989). Ideally, marketing researchers prefer *orthogonal* designs. When a linear model is fit with an orthogonal design, the parameter estimates are uncorrelated, which means each estimate is independent of the other terms in the model. More importantly, orthogonality usually implies that the coefficients will have minimum variance, though we discuss exceptions to this rule. For these reasons, orthogonal designs are usually quite good. However, for many practical problems, orthogonal designs are simply not available. In those situations, *nonorthogonal* designs must be used.

[*]This chapter is a revision of a paper that appeared in *Journal of Marketing Research*, November, 1994, pages 545–557. Warren F. Kuhfeld is now Manager, Multivariate Models R&D, SAS. Randall D. Tobias is now Manager, Linear Models R&D, SAS. Mark Garratt was Vice President, Conway | Milliken & Associates when this paper was first published in 1994 and is now with Miller Brewing Company. The authors thank Jordan Louviere, *JMR* editor Barton Weitz, and three anonymous reviewers for their helpful comments on earlier versions of this article. Thanks to Michael Ford for the idea for the second example. The *JMR* article was based on a presentation given to the AMA Advanced Research Techniques Forum, June 14, 1993, Monterey CA.

Our primary message when this paper was published in 1994 was that marketing researchers should use *D*-efficient experimental designs. This message remains as strong as ever, but today, we have much better tools for accomplishing this than we had in 1994. Most of the revisions of the original paper are due to improvements in the tools. Our new design tool, the `%MktEx` SAS macro, is easier to use than our old tools, and it usually makes better designs. Copies of this chapter (MR-2010D), the other chapters, sample code, and all of the macros are available on the Web `http://support.sas.com/resources/papers/tnote/tnote_marketresearch.html`.

Orthogonal designs are available for only a relatively small number of very specific problems. They may not be available when some combinations of factor levels are infeasible, a nonstandard number of *runs* (factor level combinations or hypothetical products) is desired, or a nonstandard model is being used, such as a model with interaction or polynomial effects. Consider the problem of designing a discrete choice study in which there are alternative specific factors, different numbers of levels within each factor, and interactions within each alternative. Orthogonal designs are not readily available for this situation, particularly when the number of runs must be limited. When an orthogonal design is not available, an alternative must be chosen—the experiment can be modified to fit some known orthogonal design, which is undesirable for obvious reasons, or a known design can be modified to fit the experiment, which may be difficult and inefficient.

Our primary purpose is to explore a third alternative, the use of optimal (or nearly optimal) designs. Such designs are typically nonorthogonal; however they are efficient in the sense that the variances and covariances of the parameter estimates are minimized. Furthermore, they are always available, even for nonstandard situations. Finding these designs usually requires the aid of a computer, but we want to emphasize that we are not advocating a black-box approach to designing experiments. Computerized design algorithms do not supplant traditional design-creation skills. Our examples show that our best designs were usually found when we used our human design skills to guide the computerized search.

First, we will summarize our main points; next, we will review some fundamentals of the design of experiments; then we will discuss computer-generated designs, a discrete-choice example, and a conjoint analysis example.

Summary of Main Points. Our goal is to explain the benefits of using computer-generated designs in marketing research. Our main points follow:

1. The goodness of an experimental design (*efficiency*) can be quantified as a function of the variances and covariances of the parameter estimates. Efficiency increases as the variances decrease. Designs should not be thought of in terms of the dichotomy between orthogonal versus nonorthogonal but rather as varying along the continuous attribute of efficiency. Some orthogonal designs are less efficient than other (orthogonal and nonorthogonal) alternatives.
2. Orthogonality is not the primary goal in design creation. It is a secondary goal, associated with the primary goal of minimizing the variances of the parameter estimates. Degree of orthogonality is an important consideration, but other factors should not be ignored.
3. For complex, nonstandard situations, computerized searches provide the only practical method of design generation for all but the most sophisticated of human designers. These situations do not have to be avoided just because it is extremely difficult to generate a good design manually.
4. The best approach to design creation is to use the computer as a tool along with traditional design skills, not as a substitute for thinking about the problem.

Background and Assumptions. We present an overview of the theory of efficient experimental design, developed for the general linear model. This topic is well known to specialists in statistical experimentation, though it is not typically taught in design classes. Then we will suggest ways in which this theory can be applied to marketing research problems.

Certain assumptions must be made before applying ordinary general linear model theory to problems in marketing research. The usual goals in linear modeling are to estimate parameters and test hypotheses about those parameters. Typically, independence and normality are assumed. In conjoint analysis, each subject rates all products and separate ordinary-least-squares analyses are run for each subject. This is not a standard general linear model; in particular, observations are not independent and normality cannot be assumed. Discrete choice models, which are nonlinear, are even further removed from the general linear model.

Marketing researchers have always made the critical assumption that designs that are good for general linear models are also good for conjoint analysis and discrete choice. We also make this assumption. Specifically, we assume the following:

1. Market share estimates computed from a conjoint analysis model using a more efficient design will be better than estimates using a less efficient design. That is, more efficient designs mean better estimates of the part-worth utilities, which lead to better estimates of product utility and market share.
2. An efficient design for a linear model is a good design for the multinomial logit (MNL) model used in discrete choice studies.

Investigating these standard assumptions is beyond the scope of this article. However, they are supported by Carson and colleagues (1994), our experiences in consumer product goods, and limited simulation results. Much more research is needed on this topic, particularly in the area of discrete choice.

Design of Experiments

Orthogonal Experimental Designs. An experimental design is a plan for running an experiment. The *factors* of an experimental design are variables that have two or more fixed values, or *levels.* Experiments are performed to study the effects of the factor levels on the dependent variable. In a conjoint or discrete-choice study, the factors are the attributes of the hypothetical products or services, and the response is preference or choice.

A simple experimental design is the *full-factorial design*, which consists of all possible combinations of the levels of the factors. For example, with five factors, two at two levels and three at three levels (denoted $2^2 3^3$), there are 108 possible combinations. In a full-factorial design, all main effects, two-way interactions, and higher-order interactions are estimable and uncorrelated. The problem with a full-factorial design is that, for most practical situations, it is too cost-prohibitive and tedious to have subjects rate all possible combinations. For this reason, researchers often use *fractional-factorial designs*, which have fewer runs than full-factorial designs. The price of having fewer runs is that some effects become confounded. Two effects are *confounded* or *aliased* when they are not distinguishable from each other.

A special type of fractional-factorial design is the *orthogonal array*, in which all estimable effects are uncorrelated. Orthogonal arrays are categorized by their *resolution.* The resolution identifies which effects, possibly including interactions, are estimable. For example, for resolution III designs, all main effects are estimable free of each other, but some of them are confounded with two-factor interactions.

For resolution V designs, all main effects and two-factor interactions are estimable free of each other. Higher resolutions require larger designs. Orthogonal arrays come in specific numbers of runs (e.g., 16, 18, 20, 24, 27, 28) for specific numbers of factors with specific numbers of levels.

Resolution III orthogonal arrays are frequently used in marketing research. The term "orthogonal array," as it is sometimes used in practice, is imprecise. It correctly refers to designs that are both orthogonal and balanced, and hence optimal. It is also imprecisely used to refer to designs that are orthogonal but not balanced, and hence potentially nonoptimal. A design is *balanced* when each level occurs equally often within each factor, which means the intercept is orthogonal to each effect. Imbalance is a generalized form of nonorthogonality, which increases the variances of the parameter estimates.

Design Efficiency. Efficiencies are measures of design goodness. Common measures of the efficiency of an $(N_D \times p)$ design matrix $\mathbf{X}$ are based on the *information matrix* $\mathbf{X}'\mathbf{X}$. The variance-covariance matrix of the vector of parameter estimates $\boldsymbol{\beta}$ in a least-squares analysis is proportional to $(\mathbf{X}'\mathbf{X})^{-1}$. An efficient design will have a "small" variance matrix, and the eigenvalues of $(\mathbf{X}'\mathbf{X})^{-1}$ provide measures of its "size." Two common efficiency measures are based on the idea of "average eigenvalue" or "average variance." *A-efficiency* is a function of the arithmetic mean of the eigenvalues, which is given by trace $((\mathbf{X}'\mathbf{X})^{-1})/p$. *D-efficiency* is a function of the geometric mean of the eigenvalues, which is given by $|(\mathbf{X}'\mathbf{X})^{-1}|^{1/p}$. A third common efficiency measure, *G-efficiency*, is based on σ_M, the maximum standard error for prediction over the candidate set. All three of these criteria are convex functions of the eigenvalues of $(\mathbf{X}'\mathbf{X})^{-1}$ and hence are usually highly correlated.

For all three criteria, if a balanced and orthogonal design exists, then it has optimum efficiency; conversely, the more efficient a design is, the more it tends toward balance and orthogonality. A design is balanced and orthogonal when $(\mathbf{X}'\mathbf{X})^{-1}$ is diagonal (for a suitably coded $\mathbf{X}$, see page 73). A design is orthogonal when the submatrix of $(\mathbf{X}'\mathbf{X})^{-1}$, excluding the row and column for the intercept, is diagonal; there may be off-diagonal nonzeros for the intercept. A design is balanced when all off-diagonal elements in the intercept row and column are zero.

These measures of efficiency can be scaled to range from 0 to 100 (for a suitably coded $\mathbf{X}$):

$$\begin{aligned}
A\text{-efficiency} &= 100 \times \frac{1}{N_D \ \text{trace}\,((\mathbf{X}'\mathbf{X})^{-1})/p} \\
D\text{-efficiency} &= 100 \times \frac{1}{N_D \ |(\mathbf{X}'\mathbf{X})^{-1}|^{1/p}} \\
G\text{-efficiency} &= 100 \times \frac{\sqrt{p/N_D}}{\sigma_M}
\end{aligned}$$

These efficiencies measure the goodness of the design relative to hypothetical orthogonal designs that may be far from possible, so they are not useful as absolute measures of design efficiency. Instead, they should be used relatively, to compare one design with another for the same situation. Efficiencies that are not near 100 may be perfectly satisfactory.

Figure 1 shows an optimal design in four runs for a simple example with two factors, using interval-measure scales for both. There are three candidate levels for each factor. The full-factorial design is shown by the nine asterisks, with circles around the optimal four design points. As this example shows,

Figure 1
Candidate Set and Optimal Design

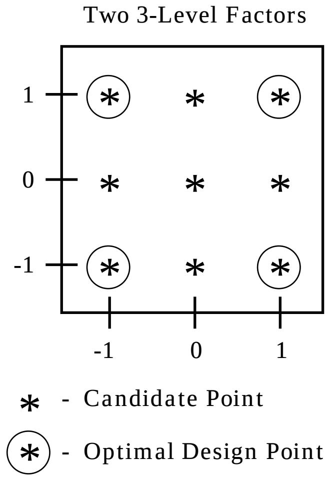


Table 1
Full-Factorial Design

Information Matrix

	Int	X1	X2	X3	-	X4	-	X5	-
Int	108	0	0	0	0	0	0	0	0
X1	0	108	0	0	0	0	0	0	0
X2	0	0	108	0	0	0	0	0	0
X3	0	0	0	108	0	0	0	0	0
-	0	0	0	0	108	0	0	0	0
X4	0	0	0	0	0	108	0	0	0
-	0	0	0	0	0	0	108	0	0
X5	0	0	0	0	0	0	0	108	0
-	0	0	0	0	0	0	0	0	108

100.0000	*D*-efficiency
100.0000	*A*-efficiency
100.0000	*G*-efficiency

efficiency tends to emphasize the corners of the design space. Interestingly, nine different sets of four points form orthogonal designs—every set of four that forms a rectangle or square. Only one of these orthogonal designs is optimal, the one in which the points are spread out as far as possible.

Computer-Generated Design Algorithms. When a suitable orthogonal design does not exist, computer-generated nonorthogonal designs can be used instead. Various algorithms exist for selecting a good set of *design points* from a set of *candidate points*. The candidate points consist of all of the factor-level combinations that can potentially be included in the design—for example the nine points in Figure 1. The number of runs, N_D, is chosen by the researcher. Unlike orthogonal arrays, N_D can be any number as long as $N_D \geq p$.[†] The algorithm searches the candidate points for a set of N_D design points that is optimal in terms of a given efficiency criterion.

It is almost never possible to list all N_D-run designs and choose *the* most efficient or optimal design, because run time is exponential in the number of candidates. For example, with $2^2 3^3$ in 18 runs, there are $108!/(18!(108-18)!) = 1.39 \times 10^{20}$ possible designs. Instead, nonexhaustive search algorithms are used to generate a small number of designs, and the most efficient one is chosen. The algorithms select points for possible inclusion or deletion, then compute rank-one or rank-two updates of some efficiency criterion. The points that most increase efficiency are added to the design. These algorithms invariably find efficient designs, but they may fail to find *the* optimal design, even for the given criterion. For this reason, we prefer to use terms like *information-efficient* and *D-efficiency* over the more common *optimal* and *D-optimal.*

There are many algorithms for generating information-efficient designs. We will begin by describing some of the simpler approaches and then proceed to the more complicated (and more reliable) algo-

[†]In fact, this restriction is not strictly necessary. So called "super-saturated" designs (Booth and Cox 1962) have more runs than parameters. However, such designs are typically not used in marketing research. The `%MktRuns` SAS macro provides some guidance on the selection of N_D. See page 1159.

rithms. Dykstra's (1971) sequential search method starts with an empty design and adds candidate points so that the chosen efficiency criterion is maximized at each step. This algorithm is fast, but it is not very reliable in finding a globally optimal design. Also, it always finds the same design (due to a lack of randomness).

The Mitchell and Miller (1970) simple exchange algorithm is a slower but more reliable method. It improves the initial design by adding a candidate point and then deleting one of the design points, stopping when the chosen criterion ceases to improve. The DETMAX algorithm of Mitchell (1974) generalizes the simple exchange method. Instead of following each addition of a point by a deletion, the algorithm makes excursions in which the size of the design may vary. These three algorithms add and delete points one at a time.

The next two algorithms add and delete points simultaneously, and for this reason, are usually more reliable for finding the truly optimal design; but because each step involves a search over all possible pairs of candidate and design points, they generally run much more slowly (by an order of magnitude). The Fedorov (1972) algorithm simultaneously adds one candidate point and deletes one design point. Cook and Nachtsheim (1980) define a modified Fedorov algorithm that finds the best candidate point to switch with each design point. The resulting procedure is generally as efficient as the simple Fedorov algorithm in finding the optimal design, but it is up to twice as fast. We extensively use one more algorithm, the coordinate exchange algorithm of Meyer and Nachtsheim (1995). This algorithm does not use a candidate set. Instead it refines an initial design by exchanging each level with every other possible level, keeping those exchanges that increase efficiency. In effect, this method uses a virtual candidate set that consists of all possible runs, even when the full-factorial candidate set is too large to generate and store.

Choice of Criterion and Algorithm. Typically, the choice of efficiency criterion is less important than the choice between manual design creation and computerized search. All of the information-efficient designs presented in this article were generated optimizing D-efficiency because it is faster to optimize than A-efficiency and because it is the standard approach. It is also possible to optimize A-efficiency, though the algorithms generally run much more slowly because the rank-one updates are more complicated with A-efficiency. G-efficiency is an interesting ancillary statistic; however, our experience suggests that attempts to maximize G-efficiency with standard algorithms do not work very well.

The candidate set search algorithms, ordered from the fastest and least reliable to the slowest and most reliable, are: sequential, simple exchange, DETMAX, and modified Fedorov. We always use the modified Fedorov and coordinate exchange algorithms even for extremely large problems; we never even try the other algorithms. For small problems in which the full factorial is no more than a few thousand runs, modified Fedorov tends to work best. For larger problems, coordinate exchange tends to be better. Our latest software, the `%MktEx` macro, tries a few iterations with both methods, then picks the best method for that problem and continues on with more iterations using just the chosen method. See page 1017 and all of the examples starting on page 285.

Nonlinear Models. The experimental design problem is relatively simple for linear models and much more complicated for nonlinear models. The usual goal when creating a design is to minimize some function of the variance matrix of the parameter estimates, such as the determinant. For linear models, the variance matrix is proportional to $(\mathbf{X}'\mathbf{X})^{-1}$, and so the design optimality problem is well-posed. However, for nonlinear models, such as the multinomial logit model used with discrete-choice data, the variance matrix depends on the true values of the parameters themselves. (See pages 265, 806, and

556 for more on efficient choice designs based on assumptions about the parameters.) Thus in general, there may not exist a design for a discrete-choice experiment that is always optimal. However, Carson and colleagues (1994) and our experience suggest that *D*-efficient designs work well for discrete-choice models.

Lazari and Anderson (1994) provide a catalog of designs for discrete-choice models, which are good for certain specific problems. For those specific situations, they may be as good as or better than computer-generated designs. However, for many real problems, cataloged designs cannot be used without modification, and modification can reduce efficiency. We carry their work one step further by discussing a general computerized approach to design generation.

Design Comparisons

Comparing Orthogonal Designs. All orthogonal designs are not perfectly or even equally efficient. In this section, we compare designs for $2^2 3^3$. Table 1 gives the information matrix, $\mathbf{X}'\mathbf{X}$, for a full-factorial design using an orthogonal coding. The matrix is a diagonal matrix with the number of runs on the diagonal. The three efficiency criteria are displayed after the information matrix. Because this is a full-factorial design, all three criteria show that the design is 100% efficient. The variance matrix (not shown) is $(1/108)\mathbf{I} = 0.0093\mathbf{I}$.

Table 2 shows the information matrix, efficiencies, and variance matrix for a classical 18-run orthogonal design for $2^2 3^3$, Chakravarti's (1956) L_{18}, for comparison with information-efficient designs with 18 runs. (The SAS ADX menu system was used to generate the design. Tables A1 and A2 contain the factor levels and the orthogonal coding used in generating Table 2.) Note that although the factors are all orthogonal to each other, X1 is not balanced. Because of this, the main effect of X1 is estimated with a higher variance (0.063) than X2 (0.056).

The precision of the estimates of the parameters critically depends on the efficiency of the experimental design. The parameter estimates in a general linear model are always unbiased (in fact, best linear unbiased [BLUE]) no matter what design is chosen. However, all designs are not equally efficient. In fact, all orthogonal designs are not equally efficient, even when they have the same factors and the same number of runs. Efficiency criteria can be used to help choose among orthogonal designs. For example, the orthogonal design in Tables 3 and A3 (from the Green and Wind 1975 carpet cleaner example) for $2^2 3^3$ is less *D*-efficient than the Chakravarti L_{18} (97.4166/98.6998 = 0.9870). The Green and Wind design can be created from a 3^5 balanced orthogonal array by collapsing two of the three-level factors into two-level factors. In contrast, the Chakravarti design is created from a $2^1 3^4$ balanced orthogonal array by collapsing only one of the three-level factors into a two-level factor. The extra imbalance makes the Green and Wind design less efficient. (Note that the off-diagonal 2 in the Green and Wind information matrix does not imply that X1 and X2 are correlated. It is an artifact of the coding scheme. The off-diagonal 0 in the variance matrix shows that X1 and X2 are uncorrelated.)

Orthogonal Versus Nonorthogonal Designs. Orthogonal designs are not always more efficient than nonorthogonal designs. Tables 4 and A4 show the results for an information-efficient, main-effects-only design in 18 runs. The OPTEX procedure of SAS software was used to generate the design, using the modified Fedorov algorithm. The information-efficient design is slightly better than the classical L_{18}, in terms of the three efficiency criteria. In particular, the ratio of the *D*-efficiencies for the classical and information-efficient designs are 99.8621/98.6998 = 1.0118. In contrast to the L_{18}, this design is

Table 2
Orthogonal Design
Information Matrix

	Int	X1	X2	X3	-	X4	-	X5	-
Int	18	6	0	0	0	0	0	0	0
X1	6	18	0	0	0	0	0	0	0
X2	0	0	18	0	0	0	0	0	0
X3	0	0	0	18	0	0	0	0	0
-	0	0	0	0	18	0	0	0	0
X4	0	0	0	0	0	18	0	0	0
-	0	0	0	0	0	0	18	0	0
X5	0	0	0	0	0	0	0	18	0
-	0	0	0	0	0	0	0	0	18

98.6998	*D*-efficiency
97.2973	*A*-efficiency
94.8683	*G*-efficiency

Variance Matrix

	Int	X1	X2	X3	-	X4	-	X5	-
Int	63	-21	0	0	0	0	0	0	0
X1	-21	63	0	0	0	0	0	0	0
X2	0	0	56	0	0	0	0	0	0
X3	0	0	0	56	0	0	0	0	0
-	0	0	0	0	56	0	0	0	0
X4	0	0	0	0	0	56	0	0	0
-	0	0	0	0	0	0	56	0	0
X5	0	0	0	0	0	0	0	56	0
-	0	0	0	0	0	0	0	0	56

Note: multiply variance matrix values by 0.001.

Table 3
Green & Wind Orthogonal Design
Information Matrix

	Int	X1	X2	X3	-	X4	-	X5	-
Int	18	-6	-6	0	0	0	0	0	0
X1	-6	18	2	0	0	0	0	0	0
X2	-6	2	18	0	0	0	0	0	0
X3	0	0	0	18	0	0	0	0	0
-	0	0	0	0	18	0	0	0	0
X4	0	0	0	0	0	18	0	0	0
-	0	0	0	0	0	0	18	0	0
X5	0	0	0	0	0	0	0	18	0
-	0	0	0	0	0	0	0	0	18

97.4166	*D*-efficiency
94.7368	*A*-efficiency
90.4534	*G*-efficiency

Variance Matrix

	Int	X1	X2	X3	-	X4	-	X5	-
Int	69	21	21	0	0	0	0	0	0
X1	21	63	0	0	0	0	0	0	0
X2	21	0	63	0	0	0	0	0	0
X3	0	0	0	56	0	0	0	0	0
-	0	0	0	0	56	0	0	0	0
X4	0	0	0	0	0	56	0	0	0
-	0	0	0	0	0	0	56	0	0
X5	0	0	0	0	0	0	0	56	0
-	0	0	0	0	0	0	0	0	56

Notes: multiply variance matrix values by 0.001.

balanced in all the factors, but X1 and X2 are slightly correlated, shown by the 2's off the diagonal. There is no *completely* orthogonal (that is, both balanced and orthogonal) $2^2 3^3$ design in 18 runs.[‡] The nonorthogonality in Table 4 has a much smaller effect on the variances of X1 and X2 (1.2%) than the lack of balance in the orthogonal design in Table 2 has on the variance of X2 (12.5%). In optimizing efficiency, the search algorithms effectively optimize both balance and orthogonality. In contrast, in some orthogonal designs, balance and efficiency may be sacrificed to preserve orthogonality.

This example shows that a nonorthogonal design may be more efficient than an unbalanced orthogonal design. We have seen this phenomenon with other orthogonal designs and in other situations as well. *Preserving orthogonality at all costs can lead to decreased efficiency.* Orthogonality was extremely important in the days before general linear model software became widely available. Today, it is more important to consider efficiency when choosing a design. These comparisons are interesting because they illustrate in a simple example how lack of orthogonality and imbalance affect efficiency. Nonorthogonal designs will never be more efficient than balanced orthogonal designs, when they exist. However, nonorthogonal designs may well be more efficient than unbalanced orthogonal designs. Although this point is interesting and important, what is most important is that good nonorthogonal designs exist in

[‡]In order for the design to be both balanced and orthogonal, the number of runs must be divisible by 2, 3, 2×2, 3×3, and 2×3. Since 18 is not divisible by 2×2, orthogonality and balance are not both simultaneously possible for this design.

Table 4
Information-Efficient Orthogonal Design
Information Matrix

	Int	X1	X2	X3	-	X4	-	X5	-
Int	18	0	0	0	0	0	0	0	0
X1	0	18	2	0	0	0	0	0	0
X2	0	2	18	0	0	0	0	0	0
X3	0	0	0	18	0	0	0	0	0
-	0	0	0	0	18	0	0	0	0
X4	0	0	0	0	0	18	0	0	0
-	0	0	0	0	0	0	18	0	0
X5	0	0	0	0	0	0	0	18	0
-	0	0	0	0	0	0	0	0	18

99.8621	*D*-efficiency
99.7230	*A*-efficiency
98.6394	*G*-efficiency

Variance Matrix

	Int	X1	X2	X3	-	X4	-	X5	-
Int	56	0	0	0	0	0	0	0	0
X1	0	56	-6	0	0	0	0	0	0
X2	0	-6	56	0	0	0	0	0	0
X3	0	0	0	56	0	0	0	0	0
-	0	0	0	0	56	0	0	0	0
X4	0	0	0	0	0	56	0	0	0
-	0	0	0	0	0	0	56	0	0
X5	0	0	0	0	0	0	0	56	0
-	0	0	0	0	0	0	0	0	56

Notes: multiply variance matrix values by 0.001.
The diagonal entries for X1 and X2 are slightly larger at 0.0563 than the other diagonal entries of 0.0556.

Table 5
Unrealistic Combinations Excluded
Information Matrix

	Int	X1	X2	X3	-	X4	-	X5	-
Int	18	0	0	0	0	0	0	0	0
X1	0	18	2	0	0	0	0	0	0
X2	0	2	18	0	0	0	0	0	0
X3	0	0	0	18	0	0	0	0	0
-	0	0	0	0	18	0	0	0	0
X4	0	0	0	0	0	18	0	-6	5
-	0	0	0	0	0	0	18	5	0
X5	0	0	0	0	0	-6	5	18	0
-	0	0	0	0	0	5	0	0	18

96.4182	*D*-efficiency
92.3190	*A*-efficiency
91.0765	*G*-efficiency

Variance Matrix

	Int	X1	X2	X3	-	X4	-	X5	-
Int	56	0	0	0	0	0	0	0	0
X1	0	56	-6	0	0	0	0	0	0
X2	0	-6	56	0	0	0	0	0	0
X3	0	0	0	56	0	0	0	0	0
-	0	0	0	0	56	0	0	0	0
X4	0	0	0	0	0	69	-7	25	-20
-	0	0	0	0	0	-7	61	-20	2
X5	0	0	0	0	0	25	-20	69	-7
-	0	0	0	0	0	-20	2	-7	61

Notes: multiply variance matrix values by 0.001.

many situations in which no orthogonal designs exist. These designs are also discussed and at a more basic level starting on page 63.

Design Considerations

Codings and Efficiency. The specific design matrix coding does not affect the relative *D*-efficiency of competing designs. Rank-preserving linear transformations are immaterial, whether they are from full-rank indicator variables to effects coding or to an orthogonal coding such as the one shown in Table A2. Any full-rank coding is equivalent to any other. The absolute *D*-efficiency values will change, but the ratio of two *D*-efficiencies for competing designs is constant. Similarly, scale for quantitative factors does not affect relative efficiency. The proof is simple. If design $\mathbf{X}_1$ is recoded to $\mathbf{X}_1\mathbf{A}$, then $|(\mathbf{X}_1\mathbf{A})'(\mathbf{X}_1\mathbf{A})| = |\mathbf{A}'\mathbf{X}_1'\mathbf{X}_1\mathbf{A}| = |\mathbf{A}\mathbf{A}'||\mathbf{X}_1'\mathbf{X}_1|$. The relative efficiency of design $\mathbf{X}_1$ compared to $\mathbf{X}_2$ is the same as $\mathbf{X}_1\mathbf{A}$ compared to $\mathbf{X}_2\mathbf{A}$, since the $|\mathbf{A}\mathbf{A}'|$'s terms in efficiency ratios will cancel. We prefer the orthogonal coding because it yields "nicer" information matrices with the number of runs on the diagonal and efficiency values scaled so that 100 means perfect efficiency.

Quantitative Factors. The factors in an experimental design are usually qualitative (nominal), but quantitative factors such as price are also important. With quantitative factors, the choice of levels depends on the function of the original variable that is modeled. To illustrate, consider a pricing study in which price ranges from \$0.99 to \$1.99. If a linear function of price is modeled, only two levels of price should be used—the end points (\$0.99 and \$1.99). Using prices that are closer together is inefficient; the variances of the estimated coefficients will be larger. The efficiency of a given design is affected by the coding of quantitative factors, even though the relative efficiency of competing designs is unaffected by coding. Consider treating the second factor of the Chakravarti L_{18}, $2^2 3^3$ as linear. It is nearly three times more D-efficient to use \$0.99 and \$1.99 as levels instead of \$1.49 and \$1.50 (58.6652/21.0832 = 2.7826). To visualize this, imagine supporting a yard stick (line) on your two index fingers (with two points). The effect on the slope of the yard stick of small vertical changes in finger locations is much greater when your fingers are closer together than when they are near the ends.

Of course there are other considerations besides the numerical measure of efficiency. It would not make sense to use prices of \$0.01 and \$1,000,000 just because that is more efficient than using \$0.99 and \$1.99. The model is almost certainly not linear over this range. To maximize efficiency, the range of experimentation for quantitative factors should be as large as possible, given that the model is plausible.

The number of levels also affects efficiency. Because two points define a line, it is inefficient to use more than two points to model a linear function. When a quadratic function is used (x and x^2 are included in the model), three points are needed—the two extremes and the midpoint. Similarly, four points are needed for a cubic function. More levels are needed when the functional form is unknown. Extra levels allow for the examination of complicated nonlinear functions, with a cost of decreased efficiency for the simpler functions. When the function is assumed to be linear, experimental points should not be spread throughout the range of experimentation. See page 1213 for a discussion of nonlinear functions of quantitative factors in conjoint analysis.

Most of the discussion outside this section has concerned qualitative (nominal) factors, even if that was not always explicitly stated. Quantitative factors complicate general design characterizations. For example, we previously stated that "if a balanced and orthogonal design exists, then it has optimum efficiency." This statement must be qualified to be absolutely correct. The design would not be optimal if, for example, a three-level factor were treated as quantitative and linear.

Nonstandard Algorithms and Criteria. Other researchers have proposed other algorithms and criteria. Steckel, DeSarbo, and Mahajan (SDM) (1991) proposed using computer-generated experimental designs for conjoint analysis to exclude unacceptable combinations from the design. They considered a nonstandard measure of design goodness based on the determinant of the (m-factor $\times$ m-factor) correlation matrix ($|\mathbf{R}|$) instead of the customary determinant of the (p-parameter $\times$ p-parameter) variance matrix ($|(\mathbf{X'X})^{-1}|$). The SDM approach represents each factor by a single column rather than as a set of coded indicator variables. Designs generated using nonstandard criteria will not generally be efficient in terms of standard criteria like A-efficiency and D-efficiency, so the parameter estimates will have larger variances. To illustrate graphically, see Figure 1. The criterion $|\mathbf{R}|$ cannot distinguish between any of the nine different four-point designs, constructed from this candidate set, that form a square or a rectangle. All are orthogonal; only one is optimal.

We generated a D-efficient design for SDM's example, treating the variables as all quantitative (as they did). The $|\mathbf{R}|$ for the SDM design is 0.9932, whereas the $|\mathbf{R}|$ for the information-efficient design is 0.9498. The SDM approach works quite well in maximizing $|\mathbf{R}|$; hence the SDM design is close to orthogonal. However, efficiency is not always maximized when orthogonality is maximized. The SDM design is approximately 75% as D-efficient as a design generated with standard criteria and algorithms ($70.1182/93.3361 = 0.7512$).

Choosing a Design. Computerized search algorithms generate many designs, from which the researcher must choose one. Often, several designs are tied or nearly tied for the best D, A, and G information efficiencies. A design should be chosen after examining the design matrix, its information matrix, its variance matrix, factor correlations, and levels frequencies. *It is important to look at the results and not just routinely choose the design from the top of the list.*

For studies involving human subjects, achieving at least nearly-balanced designs is an important consideration. Consider for example a two-level factor in an 18-run design in which one level occurs 12 times and the other level occurs 6 times versus a design in which each level occurs 9 times. Subjects who see one level more often than the other may try to read something into the study and adjust their responses in some way. Alternatively, subjects who see one level most often may respond differently than those who see the second level most often. These are not concerns with nearly balanced designs. One design selection strategy is to choose the most balanced design from the top few.

Many other strategies can be used. Perhaps correlation and imprecision are tolerable in some variables but not in others. Perhaps imbalance is tolerable, but the correlations between the factors should be minimal. Goals will no doubt change from experiment to experiment. Choosing a suitable design can be part art and part science. Efficiency should always be considered when choosing between alternative designs, even manually created designs, but it is not the only consideration.[*]

Adding Observations or Variables. These techniques can be extended to augment an existing design. A design with r runs can be created by augmenting m specified combinations (established brands or existing combinations) with $r - m$ combinations chosen by the algorithm. Alternatively, combinations that must be used for certain variables can be specified, and then the algorithm picks the levels for the other variables (Cook and Nachtsheim 1989). This can be used to ensure that some factors are balanced or uncorrelated; another application is blocking factors. Using design algorithms, we are able to establish numbers of runs and blocking patterns that fit into practical fielding schedules.

Designs with Interactions. There is a growing interest in using both main effects and interactions in discrete-choice models, because interaction and cross-effect terms may improve aggregate models (Elrod, Louviere, and Davey 1992). The current standard for choice models is to have all main-effects estimable both within and between alternatives. It is often necessary to estimate interactions within alternatives, such as in modeling separate price elasticities for product forms, sizes or packages. For certain classes of designs, in which a brand appears in only a subset of runs, it is often necessary to have estimable main-effects, own-brand interactions, and cross-effects in the submatrix of the design in which that brand is present. One way to ensure estimability is to include in the model interactions between the alternative-specific variables of interest and the indicator variables that control for presence or absence of the brand in the choice set. Orthogonal designs that allow for estimation of interactions are usually very large, whereas efficient nonorthogonal designs can be generated for any linear model, including models with interactions, and for any (reasonable) number of runs.

[*]See the `%MktEval` macro, page 1012, for a tool that helps evaluate designs.

Unrealistic Combinations. It is sometimes useful to exclude certain combinations from the candidate set. SDM (1991) have also considered this problem. Consider a discrete-choice model for several brands and their line extensions. It may not make sense to have a choice set in which the line extension is present and the "flagship" brand absent. Of course, as we eliminate combinations, we may introduce unavoidable correlation between the parameter estimates. In Tables 5 and A5, the twenty combinations where (X1 = 1 and X2 = 1 and X3 = 1) or (X4 = 1 and X5 = 1) were excluded and an 18-run design was generated with the modified Fedorov algorithm. With these restrictions, all three efficiency criteria dropped, for example 96.4182/99.8621 = 0.9655. This shows that the design with excluded combinations is almost 97% as D-efficient as the best (unrestricted) design. The information matrix shows that X1 and X2 are correlated, as are X4 and X5. This is the price paid for obtaining a design with only realistic combinations.

In the "Quantitative Factors" section, we stated "Because two points define a line, it is inefficient to use more than two points to model a linear function." When unrealistic combinations are excluded, this statement may no longer be true. For example, if minimum price with maximum size is excluded, an efficient design may involve the median price and size.

Choosing the Number of Runs. Deciding on a number of runs for the design is a complicated process; it requires balancing statistical concerns of estimability and precision with practical concerns like time and subject fatigue. Optimal design algorithms can generate designs for any number of runs greater than or equal to the number of parameters. The variances of the least-squares estimates of the part-worth utilities will be roughly inversely proportional to both the D-efficiency and the number of runs. In particular, for a given number of runs, a D-efficient design will give more accurate estimates than would be obtained with a less efficient design. A more precise value for the number of choices depends on the ratio of the inherent variability in subject ratings to the absolute size of utility that is considered important. Subject concerns probably outweigh the statistical concerns, and the best course is to provide as many products as are practical for the subjects to evaluate. In any case, the use of information-efficient designs provides more flexibility than manual methods.

Asymmetry in the Number of Levels of Variables. In many practical applications of discrete-choice modeling, there is asymmetry in the number of factor levels, and interaction and polynomial parameters must be estimated. One common method for generating choice model designs is to create a resolution III orthogonal array and modify it. The starting point is a $q^{\Sigma M_j}$ design, where q represents a fixed number of levels across all attributes and M_j represents the number of attributes for brand j. For example, in the "Consumer Food Product" example in a subsequent section, with five brands with 1, 3, 1, 2, and 1 attributes and with each attribute having at most four levels, the starting point is a 4^8 orthogonal array. Availability cross-effect designs are created by letting one of the M_j variables function as an indicator for presence/absence of each brand or by allowing one level of a common variable (price) to operate as the indicator. These methods are fairly straightforward to implement in designs in which the factor levels are all the same, but they become quite difficult to set up when there are different numbers of levels for some factors or in which specific interactions must be estimable.

Asymmetry in the number of levels of factors may be handled either by using the "coding down" approach (Addelman 1962b) or by expansion. In the coding down approach, designs are created using factors that have numbers of levels equal to the largest number required in the design. Factors that have fewer levels are created by recoding. For example, a five-level factor {1, 2, 3, 4, 5} can be recoded into a three-level factor by duplicating levels {1, 1, 2, 2, 3}. The variables will still be orthogonal because the indicator variables for the recoding are in a subspace of the original space. However, recoding introduces imbalance and inefficiency. The second method is to expand a factor at k-levels into several variables at some fraction of k-levels. For example, a four-level variable can be expanded into three orthogonal two-level variables. In many cases, both methods must be used to achieve the required design.

These approaches are difficult for a simple main-effect design of resolution III and extremely difficult when interactions between asymmetric factors must be considered. In practical applications, asymmetry is the norm. Consider for example the form of an analgesic product. One brand may have caplet and tablet varieties, another may have tablet, liquid, and chewable forms. In a discrete-choice model, these two brand/forms must be modeled as asymmetric alternative-specific factors. If we furthermore anticipated that the direct price elasticity might vary, depending on the form, we would need to estimate the interaction of a quantitative price variable with the nominal-level form variable.

Computerized search methods are simpler to use by an order of magnitude. They provide asymmetric designs that are usually nearly balanced, as well as providing easy specification for interactions, polynomials and continuous by class effects.

Strategies for Many Variables. Consider generating a 3^{15} design in 36 runs. There are 14,348,907 combinations in the full-factorial design, which is too many to use even for a candidate set. For problems like this, the coordinate exchange algorithm (Meyer and Nachtsheim 1995)] works well. The `%MktEx` macro which uses coordinate exchange with a partial orthogonal array initialization easily finds design over 98.9% D-efficient. Even designs with over 100 variables can be created this way.

Examples

Choice of Consumer Food Products. Consider the problem of using a discrete choice model to study the effect of introducing a retail food product. This may be useful, for example, to refine a marketing plan or to optimize a product prior to test market. A typical brand team will have several concerns such as knowing the potential market share for the product, examining the source of volume, and providing guidance for pricing and promotions. The brand team may also want to know what brand attributes have competitive clout and want to identify competitive attributes to which they are vulnerable.

To develop this further, assume our client wishes to introduce a line extension in the category of frozen entrees. The client has one nationally branded competitor, a regional competitor in each of three regions, and a profusion of private label products at the grocery chain level. The product comes in two different forms: stove-top or microwaveable. The client believes that the private labels are very likely to mimic this line extension and to sell it at a lower price. The client suspects that this strategy on the part of private labels may work for the stove-top version but not for the microwaveable, in which they have the edge on perceived quality. They also want to test the effect of a shelf-talker that will draw attention to their product.

This problem may be set up as a discrete choice model in which a respondent's choice among brands, given choice set C_a of available brands, will correspond to the brand with the highest utility. For each brand i, the utility U_i is the sum of a systematic component V_i and a random component e_i. The probability of choosing brand i from choice set C_a is therefore:

$$P(i|C_a) = P(U_i > \max(U_j)) = P(V_i + e_i > \max(V_j + e_j)) \;\; \forall \;\; (j \neq i) \in C_a$$

Table 6
Factors and Levels

Alternative	Factor	Levels	Brand	Description
1	X1	4	Client	3 prices + absent
2	X2	4	Client Line Extension	3 prices + absent
	X3	2		microwave/stove-top
	X4	2		shelf-talker yes/no
3	X5	3	Regional	2 prices + absent
4	X6	3	Private Label	2 prices + absent
	X7	2		microwave/stove-top
5	X8	3	Competitor	2 prices + absent

Assuming that the e_i follow an extreme value type I distribution, the conditional probabilities $P(i|C_a)$ can be found using the MNL formulation of McFadden (1974)

$$P(i|C_a) = \exp(V_i) / \sum_{j \in C_a} \exp(V_j)$$

One of the consequences of the MNL formulation is the property of independence of irrelevant alternatives (IIA). Under the assumption of IIA, all cross-effects are assumed to be equal, so that if a brand gains in utility, it draws share from all other brands in proportion to their current shares. Departures from IIA exist when certain subsets of brands are in more direct competition and tend to draw a disproportionate amount of share from each other than from other members in the category. One way to capture departures from IIA is to use the mother logit formulation of McFadden (1974). In these models, the utility for brand i is a function of both the attributes of brand i and the attributes of other brands. The effect of one brand's attributes on another is termed a *cross-effect*. In the case of designs in which only subsets C_a of the full shelf set C appear, the effect of the presence or absence of one brand on the utility of another is termed an *availability cross-effect*.

In the frozen entree example, there are five alternatives: the client, the client's line extension, a national branded competitor, a regional brand and a private label brand. Several regional and private labels can be tested in each market, then aggregated for the final model. Note that the line extension is treated as a separate alternative rather than as a "level" of the client brand. This enables us to model the source of volume for the new entry and to quantify any cannibalization that occurs. Each brand is shown at either two or three price points. Additional price points are included so that quadratic models of price elasticity can be tested. The indicator for the presence or absence of any brand in the shelf set is coded using one level of the price variable. The layout of factors and levels is given in Table 6.

In addition to intercepts and main effects, we also require that all two-way interactions within alternatives be estimable: X2*X3, X2*X4, X3*X4 for the line extension and X6*X7 for private labels. This will enable us to test for different price elasticities by form (stove-top versus microwaveable) and to see if the promotion works better combined with a low price or with different forms. Using a linear model for X1-X8, the total number of parameters including the intercept, all main effects, and two-way interactions with brand is 25. This assumes that price is treated as qualitative. The actual number of parameters in the choice model is larger than this because of the inclusion of cross-effects. Using indicator variables to code availability, the systematic component of utility for brand i can be expressed as:

$$V_i = a_i + \sum_k (b_{ik} \times x_{ik}) + \sum_{j \neq i} z_j (d_{ij} + \sum_l (g_{ijl} \times x_{jl}))$$

where

a_i	$=$ intercept for brand i
b_{ik}	$=$ effect of attribute k for brand i, where $k = 1, .., K_i$
x_{ik}	$=$ level of attribute k for brand i
d_{ij}	$=$ availability cross-effect of brand j on brand i
z_j	$=$ availability code $= \begin{cases} 1 & \text{if } j \in C_a, \\ 0 & \text{otherwise} \end{cases}$
g_{ijl}	$=$ cross-effect of attribute l for brand j on brand i, where $l = 1, .., L_j$
x_{jl}	$=$ level of attribute l for brand j.

The x_{ik} and x_{jl} might be expanded to include interaction and polynomial terms. In an availability cross-effects design, each brand is present in only a fraction of choice sets. The size of this fraction or subdesign is a function of the number of levels of the alternative-specific variable that is used to code availability (usually price). For example, if price has three valid levels and a fourth "zero" level to indicate absence, then the brand will appear in only three out of four runs. Following Lazari and Anderson (1994), the size of each subdesign determines how many model equations can be written for each brand in the discrete choice model. If X_i is the subdesign matrix corresponding to V_i, then each X_i must be full rank to ensure that the choice set design provides estimates for all parameters.

To create the design, a full candidate set is generated consisting of 3456 runs. It is then reduced to 2776 runs that contain between two and four brands so that the respondent is never required to compare more than four brands at a time. In the algorithm model specification, we designate all variables as classification variables and require that all main effects and two-way interactions within brands be estimable. The number of runs to use follows from a calculation of the number of parameters that we wish to estimate in the various submatrices $\mathbf{X}_i$ of $\mathbf{X}$. Assuming that there is a category "None" used as a reference cell, the numbers of parameters required for various alternatives are shown in the Table 7 along with the size of submatrices (rounded down) for various numbers of runs. Parameters for quadratic price models are given in parentheses. Note that the effect of private label being in a microwaveable or stove-top form (stove/micro cross-effect) is an explicit parameter under the client line extension.

The number of runs chosen was N=26. This number provides adequate degrees of freedom for the linear price model and will also allow estimation of direct quadratic price effects. To estimate quadratic cross-effects for price would require 32 runs at the very least. Although the technique of using two-way interactions between nominal level variables will usually guarantee that all direct and cross-effects are estimable, it is sometimes necessary and a good practice to check the ranks of the submatrices for more complex models (Lazari and Anderson 1994). Creating designs for cross-effects can be difficult, even with the aid of a computer.

Table 7
Parameters

Effect	Client	Client Line Extension	Regional	Private Label	Competitor
intercept	1	1	1	1	1
availability cross-effects	4	4	4	4	4
direct price effect	1 (2)	1 (2)	1	1	1
price cross-effects	4 (8)	4 (8)	4	4	4
stove versus microwave	-	1	-	1	-
stove/micro cross-effects	-	1	-	-	-
shelf-talker	-	1	-	-	-
price*stove/microwave	-	1 (2)	-	1	-
price*shelf-talker	-	1 (2)	-	-	-
stove/micro*shelf-talker	-	1	-	-	-
Total	10 (15)	16 (23)	10	12	10
Subdesign size					
22 runs	16	16	14	14	14
26 runs	19	19	17	17	17
32 runs	24	24	21	21	21

It took approximately 4.5 minutes to generate a design. The final (unrandomized) design in 26 runs is in table A6.[†] The coded choice sets are presented in Table A7 and the level frequencies are presented in Table A8. Note that the runs have been ordered by the presence/absence of the shelf-talker. This ordering is done because it is unrealistic to think that once the respondent's attention has been drawn in by the promotion, it can just be "undrawn." The two blocks that result may be shown to two groups of people or to the same people sequentially. It would be extremely difficult and time consuming to generate a design for this problem without a computerized algorithm.

Conjoint Analysis with Aggregate Interactions. This example illustrates creating a design for a conjoint analysis study. The goal is to create a 3^6 design in 90 runs. The design consists of five blocks of 18 runs each, so each subject will only have to rate 18 products. Within each block, main-effects must be estimable. In the aggregate, all main-effects and two-way interactions must be estimable. (The utilities from the main-effects models will be used to cluster subjects, then in the aggregate analysis, clusters of subjects will be pooled across blocks and the blocking factor ignored.) Our goal is to create a design that is simultaneously efficient in six ways. Each of the five blocks should be an efficient design for a first-order (main-effects) model, and the aggregate design should be efficient for the second-order (main-effects and two-way interactions) model. The main-effects models for the five blocks have $5(1 + 6(3 - 1)) = 65$ parameters. In addition, there are $(6 \times 5/2)(3 - 1)(3 - 1) = 60$ parameters for interactions in the aggregate model. There are more parameters than runs, but not all

[†]This is the design that was presented in the original 1994 paper, which due to differences in the random number seeds, is not reproduced by today's tools.

parameters will be simultaneously estimated.

One approach to this problem is the Bayesian regression method of DuMouchell and Jones (1994). Instead of optimizing $|\mathbf{X}'\mathbf{X}|$, we optimized $|\mathbf{X}'\mathbf{X}+\mathbf{P}|$, where $\mathbf{P}$ is a diagonal matrix of prior precisions. This is analogous to ridge regression, in which a diagonal matrix is added to a rank-deficient $\mathbf{X}'\mathbf{X}$ to create a full-rank problem. We specified a model with a blocking variable, main effects for the six factors, block-effect interactions for the six factors, and all two-way interactions. We constructed $\mathbf{P}$ to contain zeros for the blocking variable, main effects, and block-effect interactions, and 45s (the number of runs divided by 2) for the two-way interactions. Then we used the modified Fedorov algorithm to search for good designs.

With an appropriate coding for $\mathbf{X}$, the value of the prior precision for a parameter roughly reflects the number of runs worth of prior information available for that parameter. The larger the prior precision for a parameter, the less information about that parameter is in the final design. Specifying a nonzero prior precision for a parameter reduces the contribution of that parameter to the overall efficiency. For this problem, we wanted maximal efficiency for the within-subject main-effects models, so we gave a nonzero prior precision to the aggregated two-way interactions.

Our best design had a D-efficiency for the second-order model of 63.9281 (with a D-efficiency for the aggregate main-effects model of 99.4338) and D-efficiencies for the main-effects models within each block of 100.0000, 100.0000, 100.0000, 99.0981, and 98.0854. The design is completely balanced within all blocks. We could have specified other values in $\mathbf{P}$ and gotten better efficiency for the aggregate design but less efficiency for the blocks. Choice of $\mathbf{P}$ depends in part on the primary goals of the experiment. It may require some simulation work to determine a good choice of $\mathbf{P}$.

All of the examples in this article so far have been straight-forward applications of computerized design methodology. A set of factors, levels, and estimable effects was specified, and the computer looked for an efficient design for that specification. Simple problems, such as those discussed previously, require only a few minutes of computer time. This problem was much more difficult, so we let a work station generate designs for about 72 hours. (We could have found less efficient but still acceptable designs in much less time.) We were asking the computer to find a good design out of over 9.6×10^{116} possibilities. This is like looking for a needle in a haystack, when the haystack is the size of the entire known universe. With such problems, we may do better if we use our intuition to give the computer "hints," forcing certain structure into the design. To illustrate, we tried this problem again, this time using a different approach.

We used the modified Fedorov algorithm to generate main-effects only 3^6 designs in 18 runs. We stopped when we had ten designs all with 100% efficiency. We then wrote an ad hoc program that randomly selected five of the ten designs, randomly permuted columns within each block, and randomly permuted levels within each block. These operations do not affect the first-order efficiencies but do affect the overall efficiency for the aggregate design. When an operation increased efficiency, the new design was kept. We iterated over the entire design 20 times. We let the program run for about 16 hours, which generated 98 designs, and we found our best design in three hours. Our best design had a D-efficiency for the second-order model of 68.0565 (versus 63.9281 previously), and all first-order efficiencies of 100.

Many other variations on this approach could be tried. For example, columns and blocks could be chosen at random, instead of systematically. We performed excursions of up to eight permutations before we reverted to the previous design. This number could be varied. It seemed that permuting the levels helped more than permuting the columns, though this was not thoroughly investigated. Whatever is done, it is important to consider efficiency. For example, just randomly permuting levels can create very inefficient designs.

For this particular problem, the ad hoc algorithm generated better designs than the Bayesian method, and it required less computer time. In fact, 91 out of the 98 ad hoc designs were better than the best Bayesian design. However, the ad hoc method required much more programmer time. It is possible to manually create a design for this situation, but it would be extremely difficult and time consuming to find an efficient design without a computerized algorithm for all but the most sophisticated of human designers. The best designs were found when used both our human design skills and a computerized search. We have frequently found this to be the case.

Conclusions

Computer-generated experimental designs can provide both better and more general designs for discrete-choice and preference-based conjoint studies. Classical designs, obtained from books or computerized tables, can be good options when they exist, but they are not the only option. The time-consuming and potentially error-prone process of finding and manually modifying an existing design can be avoided. When the design is nonstandard and there are restrictions, a computer can generate a design, and it can be done quickly. In most situations, a good design can be generated in a few minutes or hours, though for certain difficult problems more time may be necessary. Furthermore, when the circumstances of the project change, a new design can again be generated quickly.

We do not argue that computerized searches for D-efficient designs are *uniformly* superior to manually generated designs. The human designer, using intuition, experience, and heuristics, can recognize structure that an optimization algorithm cannot. On the other hand, the computerized search usually does a good job, it is easy to use, and it can create a design faster than manual methods, especially for the nonexpert. Computerized search methods and the use of efficiency criteria can benefit expert designers as well. For example, the expert can manually generate a design and then use the computer to evaluate and perhaps improve its efficiency.

In nonstandard situations, simultaneous balance and orthogonality may be unobtainable. Often, the best that can be hoped for is optimal efficiency. Computerized algorithms help by searching for the most efficient designs from a potentially very large set of possible designs. Computerized search algorithms for D-efficient designs do not supplant traditional design-creation skills. Rather, they provide helpful tools for finding good, efficient experimental designs.

Table A1
Chakravarti's L_{18}, Factor Levels

X1	X2	X3	X4	X5
-1	-1	-1	-1	-1
-1	-1	0	0	1
-1	-1	1	1	0
-1	1	-1	1	0
-1	1	0	-1	-1
-1	1	1	0	1
1	-1	-1	0	0
1	-1	-1	1	1
1	-1	0	-1	0
1	-1	0	1	-1
1	-1	1	-1	1
1	-1	1	0	-1
1	1	-1	-1	1
1	1	-1	0	-1
1	1	0	0	0
1	1	0	1	1
1	1	1	-1	0
1	1	1	1	-1

Table A2
Chakravarti's L_{18}, Orthogonal Coding

X1	X2	X3	-	X4	-	X5	-
-1	-1	-1.225	-0.707	-1.225	-0.707	-1.225	-0.707
-1	-1	0.000	1.414	0.000	1.414	1.225	-0.707
-1	-1	1.225	-0.707	1.225	-0.707	0.000	1.414
-1	1	-1.225	-0.707	1.225	-0.707	0.000	1.414
-1	1	0.000	1.414	-1.225	-0.707	-1.225	-0.707
-1	1	1.225	-0.707	0.000	1.414	1.225	-0.707
1	-1	-1.225	-0.707	0.000	1.414	0.000	1.414
1	-1	-1.225	-0.707	1.225	-0.707	1.225	-0.707
1	-1	0.000	1.414	-1.225	-0.707	0.000	1.414
1	-1	0.000	1.414	1.225	-0.707	-1.225	-0.707
1	-1	1.225	-0.707	-1.225	-0.707	1.225	-0.707
1	-1	1.225	-0.707	0.000	1.414	-1.225	-0.707
1	1	-1.225	-0.707	-1.225	-0.707	1.225	-0.707
1	1	-1.225	-0.707	0.000	1.414	-1.225	-0.707
1	1	0.000	1.414	0.000	1.414	0.000	1.414
1	1	0.000	1.414	1.225	-0.707	1.225	-0.707
1	1	1.225	-0.707	-1.225	-0.707	0.000	1.414
1	1	1.225	-0.707	1.225	-0.707	-1.225	-0.707

Table A3
Green & Wind
Orthogonal Design
Example

X1	X2	X3	X4	X5
-1	-1	-1	-1	-1
-1	-1	-1	1	0
-1	-1	0	-1	-1
-1	-1	0	0	1
-1	-1	0	1	-1
-1	-1	1	-1	0
-1	-1	1	0	1
-1	-1	1	1	0
-1	1	-1	1	1
-1	1	-1	-1	1
-1	1	0	0	0
-1	1	1	0	-1
1	-1	-1	0	-1
1	-1	-1	0	0
1	-1	0	1	1
1	-1	1	-1	1
1	1	1	1	-1
1	1	0	-1	0

Table A4
Information-Efficient
Design,
Factor Levels

X1	X2	X3	X4	X5
-1	-1	-1	0	-1
-1	-1	0	-1	0
-1	-1	0	1	-1
-1	-1	1	0	1
-1	-1	1	1	1
-1	1	-1	-1	0
-1	1	-1	0	-1
-1	1	0	-1	1
-1	1	1	1	0
1	-1	-1	-1	1
1	-1	-1	1	0
1	-1	0	0	0
1	-1	1	-1	-1
1	1	-1	1	1
1	1	0	0	1
1	1	0	1	-1
1	1	1	-1	-1
1	1	1	0	0

Table A5
Information-Efficient
Design, Unrealistic
Combinations Excluded

X1	X2	X3	X4	X5
-1	-1	-1	1	0
-1	-1	-1	-1	1
-1	-1	-1	0	-1
-1	-1	0	-1	1
-1	-1	0	0	0
-1	1	1	1	0
-1	1	1	-1	-1
-1	1	1	0	1
-1	1	0	1	-1
1	-1	1	1	-1
1	-1	1	-1	0
1	-1	1	0	1
1	-1	0	1	-1
1	1	-1	1	0
1	1	-1	-1	-1
1	1	-1	0	1
1	1	0	-1	1
1	1	0	0	0

Table A6
Consumer Food Product (Raw) Design

X1	X2	X3	X4	X5	X6	X7	X8
1	1	2	1	1	2	1	3
1	2	2	1	2	3	1	2
1	4	1	1	1	3	1	3
2	2	1	1	3	2	1	1
2	3	2	1	2	2	2	3
2	4	2	1	3	3	2	2
3	1	1	1	3	2	2	2
3	3	2	1	3	1	2	1
3	4	2	1	2	1	1	1
4	1	1	1	2	3	2	1
4	1	2	1	3	3	1	1
4	2	2	1	1	2	2	3
4	3	1	1	1	1	1	2
1	3	1	2	3	2	2	1
1	3	2	2	3	1	1	3
1	4	2	2	1	1	2	1
2	1	1	2	1	3	1	1
2	2	2	2	3	2	1	1
2	3	1	2	2	1	2	3
2	4	1	2	3	1	1	2
3	1	2	2	2	3	2	2
3	2	1	2	1	3	2	3
3	4	2	2	2	3	1	3
4	1	1	2	3	2	1	3
4	2	1	2	2	1	2	2
4	3	2	2	1	2	1	2

Table A7
Consumer Food Product Choice Set

Block 1: Shelf-Talker Absent For Client Line Extension					
Choice Set	Client Brand	Client Line Extension	Regional Brand	Private Label	National Competitor
1	$1.29	$1.39/stove	$1.99	$2.29/micro	N/A
2	$1.29	$1.89/stove	$2.49	N/A	$2.39
3	$1.29	N/A	$1.99	N/A	N/A
4	$1.69	$1.89/micro	N/A	$2.29/micro	$1.99
5	$1.69	$2.39/stove	$2.49	$2.29/stove	N/A
6	$1.69	N/A	N/A	N/A	$2.39
7	$2.09	$1.39/micro	N/A	$2.29/stove	$2.39
8	$2.09	$2.39/stove	N/A	$1.49/stove	$1.99
9	$2.09	N/A	$2.49	$1.49/micro	$1.99
10	N/A	$1.39/micro	$2.49	N/A	$1.99
11	N/A	$1.39/stove	N/A	N/A	$1.99
12	N/A	$1.89/stove	$1.99	$2.29/stove	N/A
13	N/A	$2.39/micro	$1.99	$1.49/micro	$2.39

Block 2: Shelf-Talker Present For Client Line Extension					
Choice Set	Client Brand	Client Line Extension	Regional Brand	Private Label	National Competitor
14	$1.29	$2.39/micro	N/A	$2.29/stove	$1.99
15	$1.29	$2.39/stove	N/A	$1.49/micro	N/A
16	$1.29	N/A	$1.99	$1.49/stove	$1.99
17	$1.69	$1.39/micro	$1.99	N/A	$1.99
18	$1.69	$1.89/stove	N/A	$2.29/micro	$1.99
19	$1.69	$2.39/micro	$2.49	$1.49/stove	N/A
20	$1.69	N/A	N/A	$1.49/micro	$2.39
21	$2.09	$1.39/stove	$2.49	N/A	$2.39
22	$2.09	$1.89/micro	$1.99	N/A	N/A
23	$2.09	N/A	$2.49	N/A	N/A
24	N/A	$1.39/micro	N/A	$2.29/micro	N/A
25	N/A	$1.89/micro	$2.49	$1.49/stove	$2.39
26	N/A	$2.39/stove	$1.99	$2.29/micro	$2.39

Table A8
Consumer Food Product Design Level Frequencies

Level	X1	X2	X3	X4	X5	X6	X7	X8
1	6	7	12	13	8	8	14	9
2	7	6	14	13	8	9	12	8
3	6	7			10	9		9
4	7	6						

Table A9
Consumer Food Product Design Creation Code

```
*---------------------------------------------------*
|                Construct the Design.              |
*---------------------------------------------------;

%macro bad;
  bad = (x1 < 4) + (x2 < 4) + (x5 < 3) + (x6 < 3) + (x8 < 3);
  bad = abs(bad - 3) * ((bad < 2) | (bad > 4));
  %mend;

%mktex(4 4 2 2 3 3 2 3, n=26, interact=x2*x3 x2*x4 x3*x4 x6*x7,
       restrictions=bad, outr=sasuser.choicdes)

*---------------------------------------------------*
|                  Print the Design.                |
*---------------------------------------------------;

proc format;
   value yn    1 = 'No'    2 = 'Talker';
   value micro 1 = 'Micro' 2 = 'Stove';
   run;

data key;
   missing N;
   input x1-x8;
   format x1 x2 x5 x6 x8 dollar5.2
          x4 yn. x3 x7 micro.;
   label x1 = 'Client Brand'
         x2 = 'Client Line Extension'
         x3 = 'Client Micro/Stove'
         x4 = 'Shelf Talker'
         x5 = 'Regional Brand'
         x6 = 'Private Label'
         x7 = 'Private Micro/Stove'
         x8 = 'National Competitor';
   datalines;
1.29 1.39 1 1 1.99 1.49 1 1.99
1.69 1.89 2 2 2.49 2.29 2 2.39
2.09 2.39 . . N    N    .    N
N    N    . . .    .    .    .
;

%mktlab(data=sasuser.choicdes, key=key)

proc sort out=sasuser.finchdes; by x4; run;

proc print label; id x4; by x4; run;
```

A General Method for Constructing Efficient Choice Designs

Klaus Zwerina

Joel Huber

Warren F. Kuhfeld

Abstract

Researchers have traditionally built choice designs using extensions of concepts from the general linear model design literature. We show that a computerized search strategy can generate efficient choice designs with standard personal computers. This approach holds three important advantages over previous design strategies. First, it allows the incorporation of anticipated model parameters, thereby increasing design efficiency and considerably reducing the number of required choices. Second, complex choice designs can be easily generated, allowing researchers to conduct choice experiments that more closely mirror actual market conditions. Finally, researchers can explore model and design modifications and examine trade-offs between a design's statistical benefits and its operational and behavioral costs.*

Introduction

Discrete choice experiments are becoming increasingly popular in marketing, economics, and transportation. These experiments enable researchers to model choice in an explicit competitive context, thus realistically emulating market decisions. A choice design consists of choice sets composed of several alternatives, each defined as combinations of different attribute levels. A good choice design is efficient, meaning that the parameters of the choice model are estimated with maximum precision.

A number of methods have been suggested for building choice designs (Anderson and Wiley 1992, Bunch, Louviere, and Anderson 1996, Krieger and Green 1991, Kuhfeld 2010 (page 285), Lazari and Anderson 1994, Louviere and Woodworth 1983). Most of the methods use extensions of standard linear experimental designs (Addelman 1962b, Green 1974). However, the use of linear model designs in choice experiments may be nonoptimal due to two well-known differences between linear and choice

*Klaus Zwerina is a consultant at BASF AG, Ludwigshafen, Germany. Joel Huber is Professor of Marketing, Fuqua School of Business, Duke University. Warren F. Kuhfeld is Manager, Multivariate Models R&D, Statistical Research and Development, SAS Institute Inc. We would like to thank Jim Bettman and Richard Johnson for their helpful comments on an earlier version of this chapter. Copies of this chapter (MR-2010E), the other chapters, sample code, and all of the macros are available on the Web `http://support.sas.com/resources/papers/tnote/tnote_marketresearch.html`.

models. First, probabilistic choice models are nonlinear in the parameters, implying that the statistical efficiency of a choice design depends on an (unknown) parameter vector. This property implies the need to bring anticipated parameter values in choice designs. Second, choice design efficiency depends both on the creation of appropriate profiles and properly placing them into several choice sets. For example, in a linear model design, the order of the 16 profiles in a conjoint exercise does not affect its formal efficiency, whereas the efficiency of the same 16 profiles broken into four choice sets depends critically on the grouping. Despite its limitations, linear model design theory has been used to produce satisfactory choice designs for many years, drawing on readily available tables and processes. Such carefully selected linear model designs are reasonable, general-purpose choice designs, but are generally not optimal in a statistical sense.

We present a general strategy for the computerized construction of efficient choice designs. This contribution can be viewed as an extension of the work of Kuhfeld, Tobias, and Garratt (1994) and of Huber and Zwerina (1996). Kuhfeld et al. recommended using a search algorithm to find efficient *linear* model designs. Huber and Zwerina show how to modify *choice* designs using anticipated model parameters in order to improve design efficiency. We adapt the optimization procedure outlined in Kuhfeld et al. to the principles of choice design efficiency described by Huber and Zwerina. Our approach holds several important advantages over previous choice design strategies. It (1) optimizes the "correct" criterion of minimizing estimation error rather than following linear model design principles, (2) it can generate choice designs that accommodate any anticipated parameter vector, (3) it can accommodate virtually any level of model complexity, and finally (4) it can be built using widely available software. To illustrate, we include a SAS/IML program that generates relatively simple choice designs. This program can be easily generalized to handle far more complex problems.

The chapter begins with a review of criteria for efficient choice designs and illustrates how they can be built with a computer. Then, beginning with simple designs, we illustrate how the algorithm works and how our linear model design intuition must be changed when coping with choice designs. Next, we generate more complex choice designs and show how to evaluate the impact on efficiency of different design and model modifications. We conclude with a discussion of the proposed choice design approach and directions for future research.

Criteria For Choice Design Efficiency

Measure Of Choice Design Efficiency. First, we derive a measure of efficiency in choice designs from the well-known multinomial logit model (McFadden 1974). This model assumes that consumers make choices among alternatives that maximize their perceived utility, u, given by

$$u = \mathbf{x}_i \boldsymbol{\beta} + e \qquad (1)$$

where $\mathbf{x}_i$ is a row vector of attributes characterizing alternative i, $\boldsymbol{\beta}$ is a column vector of K weights associated with these attributes, and e is an error term that captures unobserved variations in utility. Suppose that there are N choice sets, C_n, indexed by $n = 1, 2, \ldots, N$, where each choice set is characterized by a set of alternatives $C_n = \{x_{1n}, K, x_{J_n n}\}$. If the errors, e, are independently and identically Gumbel distributed, then it can be shown that the probability of choosing an alternative i from a choice set C_n is

$$P_{in}(\mathbf{X_n}, \boldsymbol{\beta}) = \frac{e^{\mathbf{x}_{in}\boldsymbol{\beta}}}{\sum_{j=1}^{J_n} e^{\mathbf{x}_{jn}\boldsymbol{\beta}}} \tag{2}$$

where $\mathbf{X_n}$ is a matrix that consists of J_n row vectors, each describing the characteristics of the alternatives, $\mathbf{x}_{jn}$. The vertical concatenation of the $\mathbf{X_n}$ matrices is called a choice design matrix $\mathbf{X}$.

The task of the analyst is to find a parameter estimate for $\boldsymbol{\beta}$ in Equation (2) that maximizes the likelihood given the data. Under very general conditions, the maximum likelihood estimator is consistent and asymptotically normal with covariance matrix

$$\boldsymbol{\Sigma} = (\mathbf{Z}'\mathbf{P}\mathbf{Z})^{-1} = \left[\sum_{n=1}^{N}\sum_{j=1}^{J_N} \mathbf{z}'_{jn} P_{jn} \mathbf{z}_{jn}\right]^{-1} \tag{3}$$

$$\text{where} \quad \mathbf{z}_{jn} = \mathbf{x}_{jn} - \sum_{i=1}^{J_n} \mathbf{x}_{in} P_{in} \ .$$

Equation (3) reveals some important properties of (nonlinear) choice models. In linear models, centering occurs across all profiles whereas in choice models, centering occurs within choice sets. This shows that in choice designs both the profile selection and the assignment of profiles to choice sets affects the covariance matrix. Moreover, in linear models, the covariance matrix does not depend on the true parameter vector, whereas in choice models the probabilities, P_{jn}, are functions of $\boldsymbol{\beta}$ and hence the covariance matrix. Assuming $\boldsymbol{\beta} = \mathbf{0}$ simplifies the design problem, however Huber and Zwerina (1996) recently demonstrated that this assumption may be costly. They showed that incorrectly assuming that $\boldsymbol{\beta} = \mathbf{0}$ may require from 10% to 50% more respondents than those built from reasonably anticipated parameters.

The goal in choice designs is to define a group of choice sets, given the anticipated $\boldsymbol{\beta}$, that minimizes the "size" of the covariance matrix, $\boldsymbol{\Sigma}$, defined in Equation (3). There are various summary measures of error size that can be derived from the covariance matrix (see, e.g., Raktoe, Hedayat, and Federer 1981). Perhaps the most intuitive summary measure is the average variance around the estimated parameters of a model. This measure is referred to in the literature as *A*-efficiency or its inversely related counterpart,

$$A - \mathit{error} = \text{trace}\,(\boldsymbol{\Sigma})/K \tag{4}$$

where K is the number of parameters. Two problems with this measure limit its suitability as an overall measure of design efficiency. First, relative *A*-error is not invariant under (nonsingular) recodings of the design matrix, i.e., design efficiency depends on the type of coding. Second, it is computationally expensive to update. A related measure,

$$D - \mathit{error} = |\boldsymbol{\Sigma}|^{1/K} \tag{5}$$

is based on the determinant as opposed to the trace of the covariance matrix. *D*-error is computationally efficient to update, and the ratios of *D*-errors are invariant under different codings of the design matrix. Since *A*-error is the arithmetic mean and *D*-error is the geometric mean of the eigenvalues of $\boldsymbol{\Sigma}$, they are generally highly correlated. D-error thereby provides a reasonable way to find designs that are "good" on alternative criteria. For example, if A-error is the ultimate criterion, we can first minimize D-error and then select the design with minimum A-error rather than minimizing A-error directly. For these reasons, D-error (or its inverse, D-efficiency or D-optimality) is the most common criterion for evaluating linear model designs and we advocate it as a criterion for choice designs.

Next, we discuss four principles of choice design efficiency defined by Huber and Zwerina (1996). Choice designs that satisfy these principles are optimal, however, these principles are only satisfied for a few special cases and under quite restrictive assumptions. The principles of orthogonality and balance that figured so prominently in linear model designs remain important in understanding what makes a good choice design, but they will be less useful in generating one.

Four Principles Of Efficient Choice Designs. Huber and Zwerina (1996) identify four principles which when jointly satisfied indicate that a design has minimal D-error. These principles are orthogonality, level balance, minimal overlap, and utility balance. *Orthogonality* is satisfied when the levels of each attribute vary independently of one another. *Level balance* is satisfied when the levels of each attribute appear with equal frequency. *Minimal overlap* is satisfied when the alternatives within each choice set have nonoverlapping attribute levels. *Utility balance* is satisfied when the utilities of alternatives within choice sets are the same, i.e., the design will be more efficient as the expected probabilities within a choice set C_n among J_n alternatives approach $1/J_n$.

These principles are useful in understanding what makes a choice design efficient, and improving any principle, holding the others constant, improves efficiency. However, for most combinations of attributes, levels, alternatives, and assumed parameter vectors, it is impossible to create a design that satisfies these principles. The proposed approach does not build choice designs from these formal principles, but instead uses a computer to directly minimize D-error. As a result, these principles may only be approximately satisfied in our designs, but they will generally be more efficient than those built directly from the principles.

A General Method For Efficient Choice Designs

Figure 1 provides a flowchart of the proposed design approach. The critical aspect of this approach involves an adaptation of Cook and Nachtsheim's (1980) modification of the Fedorov (1972) algorithm that has successfully been used to generate efficient *linear* model designs (e.g., Cook and Nachtsheim 1980, Kuhfeld et al. 1994). We will first describe the proposed choice design approach conceptually and then define the details in a context of a particular search.

The process begins by building a *candidate set*, which is a list of potential alternatives. A random selection of these alternatives is the *starting design*. The algorithm alters the starting design by exchanging its alternatives with the candidate alternatives. The algorithm finds the best exchange (if one exists) for the first alternative in the starting design. The first iteration is completed once the algorithm has sequentially found the best exchanges for all of the alternatives in the starting design. After that, the process moves back to the first alternative and continues until no substantial efficiency improvement is possible. To avoid poor local optima, the whole process can be restarted with different random starting designs and the most efficient design is selected. For example, if there are 300 alternatives in the candidate set and 50 alternatives in the choice design, then each iteration requires testing 15,000 possible exchanges, which is a reasonable problem on today's desktop computers and workstations. While there is no guarantee that it will converge to an optimal design, our experience with relatively small problems suggests that the algorithm works very well.

To illustrate the process we first generate choice designs for simple models that reveal the characteristics of efficient choice designs. In examining these simple designs, our focus is on the benefits and the insights that derive from using this approach. Then, we apply the approach to more complex design problems, such as alternative-specific designs and designs with constant alternatives. As we illustrate more complex designs, we will focus on the use of the approach, *per se*. We provide illustrative computer code in the appendix.

Choice Design Applications

Generic Models. The simplest choice models involve alternatives described by generic attributes. The utility functions for these models consist of attribute parameters that are the same for all alternatives, for example, a common price slope across all alternatives. Generic designs are appealing because they are simple and analogous to main-effects conjoint experiments. Bunch et al. (1996) evaluate ways to generate generic choice designs and show that shifted or cyclic designs generally have superior efficiency compared with other strategies for generating main effects designs. These shifted designs use an orthogonal fractional factorial to provide the "seed" alternatives for each choice set. Subsequent alternatives within a choice set are cyclically generated. The attribute levels of the new alternatives add one to the level of the previous alternative until it is at its highest level, at which point the assignment re-cycles to the lowest level.

For certain families of plans and assuming that all coefficients are zero, these shifted designs satisfy all four principles, and thus are optimal.[†] For example, consider a choice experiment with three attributes, each at three levels, defining three alternatives in each of nine choice sets. The left-hand panel of Table 1 shows a plan using the Bunch et al. (1996) method.

In this special case, all four efficiency principles are perfectly satisfied. Level balance is satisfied since each level occurs in precisely 1/3 of the cases, and orthogonality can be confirmed by noting the all pairs of attribute levels occur in precisely 1/9 of the attributes (Addelman 1962b). There is perfect minimal overlap since each level occurs exactly once in each choice set, and finally, utility balance is trivially satisfied with the assumption that $\boldsymbol{\beta} = \mathbf{0}$. More formally, it is useful to examine the covariance matrix of the (effects-coded) parameters, reported in the first panel of Table 2. The equal variances across attributes and the zero covariances across attributes both indicate optimality.

A simple design such as this could have been built from our algorithm, although using a standard orthogonal array and cyclic permutations ensured optimality. Our next example, encompassing a model with just one interaction term, illustrates the case when a computerized search is very useful in finding a statistically efficient design.

Estimating an A×B Interaction. For the previous example with nine choice sets, let us assume that the researcher is confident that there are no A×C or B×C interactions, but the A×B interaction must be estimated. The middle panel of Table 1 shows the best design we were able to find which includes this one interaction. Note that in this design, the principle of minimal overlap on attributes A and B is violated, in that attribute levels are frequently repeated within a set. In general, interactions *require* overlap of attribute levels to produce the contrasts necessary to estimate the effects.

[†]We were not able to analytically prove this, but after examining scores of designs, we have never found more efficient designs than those that satisfy all four principles.

Figure 1
Flowchart of Algorithm for Constructing Efficient Choice Designs

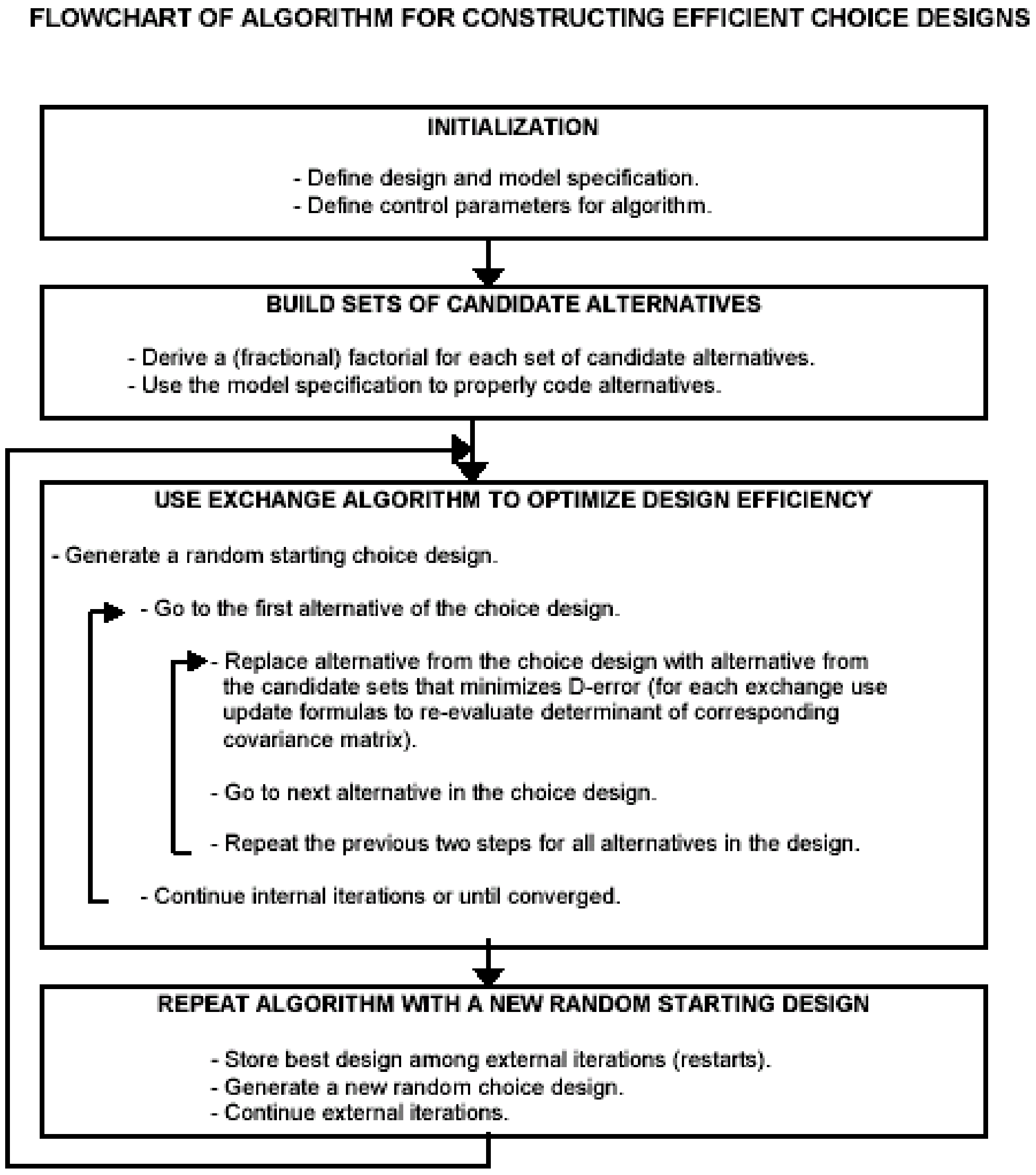

FLOWCHART OF ALGORITHM FOR CONSTRUCTING EFFICIENT CHOICE DESIGNS
INITIALIZATION
- Define design and model specification.
- Define control parameters for algorithm.
BUILD SETS OF CANDIDATE ALTERNATIVES
- Derive a (fractional) factorial for each set of candidate alternatives.
- Use the model specification to properly code alternatives.
USE EXCHANGE ALGORITHM TO OPTIMIZE DESIGN EFFICIENCY
- Generate a random starting choice design.
- Go to the first alternative of the choice design.
- Replace alternative from the choice design with alternative from the candidate sets that minimizes D-error (for each exchange use update formulas to re-evaluate determinant of corresponding covariance matrix).
- Go to next alternative in the choice design.
- Repeat the previous two steps for all alternatives in the design.
- Continue internal iterations or until converged.
REPEAT ALGORITHM WITH A NEW RANDOM STARTING DESIGN
- Store best design among external iterations (restarts).
- Generate a new random choice design.
- Continue external iterations.

Table 1
Main Effects and A×B-Interaction Effects Choice Design

$\boldsymbol{\beta}_0$-Efficient Main-Effects Design ($\boldsymbol{\beta}_0$=0 0 0 0 0 0)					$\boldsymbol{\beta}_0$-Efficient Interaction-Effects ($\boldsymbol{\beta}_0$=0 0 0 0 0 0 0 0 0)				$\boldsymbol{\beta}_1$-Efficient Interaction-Effects ($\boldsymbol{\beta}_1$=-1 0 -1 0 -1 0 0 0 0)			
Set	Alt	A	B	C	A	B	C	p($\boldsymbol{\beta}_1$)	A	B	C	p($\boldsymbol{\beta}_1$)
1	I	1	1	1	2	3	2	.495	2	1	3	.422
	II	2	2	2	2	2	3	.495	1	3	2	.422
	III	3	3	3	1	1	1	.009	3	1	1	.155
2	I	1	2	2	3	1	1	.155	3	2	2	.422
	II	2	3	3	2	2	2	.422	2	1	3	.155
	III	3	1	1	1	2	3	.422	3	3	1	.422
3	I	1	3	3	1	1	2	.042	2	2	3	.155
	II	2	1	1	1	3	1	.114	3	3	2	.422
	III	3	2	2	3	1	3	.844	2	3	3	.422
4	I	2	1	3	2	1	2	.018	2	1	2	.422
	II	3	2	1	3	3	3	.965	1	1	3	.422
	III	1	3	2	2	2	1	.018	1	2	1	.155
5	I	2	2	1	1	3	3	.245	3	1	2	.422
	II	3	3	2	3	3	2	.665	1	1	3	.155
	III	1	1	3	2	3	1	.090	3	2	1	.422
6	I	2	3	2	2	1	3	.468	1	3	3	.422
	II	3	1	3	1	2	1	.063	2	3	2	.422
	III	1	2	1	1	3	2	.468	3	2	1	.155
7	I	3	1	2	3	2	3	.665	1	2	1	.212
	II	1	2	3	3	3	1	.245	2	2	1	.576
	III	2	3	1	3	1	2	.090	1	1	2	.212
8	I	3	2	3	1	2	2	.042	1	2	3	.576
	II	1	3	1	2	3	3	.844	1	3	1	.212
	III	2	1	2	3	2	1	.114	3	1	1	.212
9	I	3	3	1	1	1	3	.114	2	2	2	.212
	II	1	1	2	2	1	1	.042	3	1	3	.576
	III	2	2	3	3	2	2	.844	2	3	1	.212
					avemaxp = .690				avemaxp = .474			
D-error($\boldsymbol{\beta}_0$) = .192					D-error($\boldsymbol{\beta}_0$) = .306				D-error($\boldsymbol{\beta}_0$) = .365			
					D-error($\boldsymbol{\beta}_1$) = .630				D-error($\boldsymbol{\beta}_1$) = .399			

The covariance matrix of this design, depicted in the lower half of Table 2, highlights the effects of incorporating the A×B interaction. Violating minimal overlap permits the estimation of the A×B interaction by sacrificing efficiency on the main effects of attribute A and B, reflected in higher variances of the main effects estimates (a1, a2, b1, and b2). The D-error of the main effect estimates increases by 24%, from .192 to .239, and the covariances across attributes A and B are no longer zero. Note also that the errors around attribute C are unchanged—they are unaffected by the A×B interaction, indicating that the algorithm was able to find a design that allowed the A×B interaction to be uncorrelated with C.

Table 2
Covariance Matrix of Main Effects and A×B-Interaction
Effects Choice Design

$\boldsymbol{\beta}_0$-Efficient Main Effects Design

	a1	a2	b1	b2	c1	c2
a1	.222	-.111	.000	.000	.000	.000
a2	-.111	.222	.000	.000	.000	.000
b1	.000	.000	.222	-.111	.000	.000
b2	.000	.000	-.111	.222	.000	.000
c1	.000	.000	.000	.000	.222	-.111
c1	.000	.000	.000	.000	-.111	.222

D-error($\boldsymbol{\beta}_0$)=.192

$\boldsymbol{\beta}_0$-Efficient Interaction Effects Design

	a1	a2	b1	b2	c1	c2	ab11	ab12	ab21	ab22
a1	.296	-.130	.019	-.019	.000	.000	-.037	.000	.000	-.019
a2	-.130	.296	-.019	.019	.000	.000	.037	.000	.000	.019
b1	.019	-.019	.296	-.130	.000	.000	.019	-.056	.000	.037
b2	-.019	.019	-.130	.296	.000	.000	-.019	.056	.000	-.037
c1	.000	.000	.000	.000	.222	-.111	.000	.000	.000	.000
c2	.000	.000	.000	.000	-.111	.222	.000	.000	.000	.000
ab11	-.037	.037	.019	-.019	.000	.000	.630	-.333	-.333	.148
ab12	.000	.000	-.056	.056	.000	.000	-.333	.556	.167	-.278
ab21	.000	.000	.000	.000	.000	.000	-.333	.167	.667	-.333
ab22	-.019	.019	.037	-.037	.000	.000	.148	-.278	-.333	.630

D-error($\boldsymbol{\beta}_0$) of main effects = .239
D-error($\boldsymbol{\beta}_0$) of all effects = .306

There are several important lessons from this simple example. First, it illustrates that a design that is "perfect" for one model may be far from optimal for a slightly different model. Adding one interaction strongly altered the covariance matrix, so efficient designs generally violate the formal principles. Second, the example shows that estimating new interactions is not without cost; being able to estimate one interaction increased by 24% the error on the main effects. Finally, the trade-off of efficiency with estimability demonstrates one of the primary benefits of this approach—it allows the analyst to understand the efficiency implications of changes in the design structure and/or model specification. This use of the approach will be illustrated again in the context of more complex choice designs.

The Impact Of Non-Zero Betas. The preceding discussion has assumed that the true parameters are zero. This assumption is justified when there is very little information about the model parameters; however, typically the analyst has some information about the relative importance of attributes or the relative value of their levels (Huber and Zwerina 1996). To show the potential gain that can come from nonzero parameters, assume that the anticipated partworths of the main effects for the three level attributes discussed previously are not 0, 0, 0, but -1, 0, 1, while the A×B-interaction effect continues to have zero parameters.[‡] Calling the new parameter vector $\boldsymbol{\beta}_1$ to distinguish it from the zero parameter

[‡]We assume for simplicity that the interaction has parameter values of zero. Note, this also produces minimal variance of estimates around zero, implying greatest power of a test in the region of that null hypothesis.

Table 3
Attributes/Levels for an Alternative-Specific Choice Experiment

	Alternative-Specific Levels		
Attributes	Coke	Pepsi	RC Cola
Price per case	\$5.69	\$5.39	\$4.49
	\$6.89	\$5.99	\$5.39
	\$7.49	\$6.59	\$5.99
Container	12 oz cans	12 oz cans	12 oz cans
	10 oz bottle	10 oz bottle	16 oz bottle
	16 oz bottle	18 oz bottle	22 oz bottle
Flavor	Regular	Regular	Regular
	Cherry Coke	Pepsi Lite	Cherry
	Diet Coke	Diet Pepsi	Diet

vector, $\boldsymbol{\beta}_0$, the third panel of Table 1 displays the efficient design using these parameters. This new design has a D-error($\boldsymbol{\beta}_1$) of 0.399. However, if instead we had used the design in the center panel, its error given $\boldsymbol{\beta}_1$ is true would have been .630, implying that 37% (1 - .399/.630) fewer respondents are needed for the "utility balanced" over the "utility neutral" design.

Comparing the last two panels in Table 1 reveals how the algorithm used the anticipated nonzero parameters to produce a more efficient design. As an index of utility balance, we calculated the average of the maximum within-choice-set choice probabilities (avemaxp). The smaller this index the harder is the average choice task and the greater is "utility balance." We can see, by using $\boldsymbol{\beta}_1$, the new design is more utility balanced than the previous design, which results in an average maximum probability of .474 compared with one of .690. We also see that the increase in utility balance sacrifices somewhat the three formal principles, reflected in an increase of D-error($\boldsymbol{\beta}_0$) from .306 to .365. The new design does not have perfect orthogonality, level balance, utility balance, or minimal overlap, but it is more efficient than any design that is perfect on any of those criteria.

More Complex Choice Designs. The proposed algorithm is very general and can be applied to virtually any level of design complexity. We will use it next to generate an alternative-specific choice design, which has a separate set of parameters for each alternative. Suppose, the researcher is interested in simulating the market behavior of three brands, Coke, Pepsi, and RC Cola, with the attribute combinations shown in Table 3.

This kind of choice experiment, which we call a market emulation study, is quite different from the generic choice design presented previously. In a market emulation study, emphasis is on predicting the impact of brand, flavor, and container decisions in the context of a realistic market place offering. What this kind of study gains in realism, it loses in the interpretability of its results. For example, since each brand only occurs at specific prices, it is much harder to disentangle the independent effects of brand and price. These designs are, however, useful in assessing the managerially critical question of the impact of, say, a 60 cent drop in the price of Coke's 16 ounce case in a realistic competitive configuration.

Since we assume that the impact of price depends on the brand to which it is attached, it is important that the impact of price be estimable within each brand.[§] Further, let us assume that the reaction to price additionally depends on the number of ounces, so that it is necessary to estimate the brand×price×container interaction. Using standard ANOVA-coding, these assumptions require four main effects (brand, flavor, container, and price for 8 *df*), four two-way interactions (brand×price, brand×flavor, brand×container, and price×container for 16 *df*), and one three-way interaction (brand×price×container for 8 *df*), resulting in a total of 32 parameters.[¶]

Suppose we want to precisely estimate these effects with a choice design consisting of 27 choice sets each composed of three alternatives.[*] The candidate set of alternatives comprises the $3^4 = 81$ possible alternatives, and the initial design is a random selection from these. The algorithm exchanges alternatives between the candidate set and the starting design until the efficiency gain becomes negligible. In the example with 27 choice sets and 32 parameters, *D*-error is .167. This statistic provides a baseline for evaluating other related designs, which we will generate in the following section.

Evaluating Design Modifications. The proposed approach can be used to evaluate design modifications. Typically, efficiency is meaningful within a relatively narrow family of designs, limited to a particular attribute structure, model specification, and number of alternatives per choice set. For many applications, optimizing a design within such a narrow design family is too restrictive. Most analysts are not tightly bound to a particular number of alternatives per choice set or even particular attributes, but are interested in exploring the impact of changes in these specifications on the precision of the parameter estimates. We will demonstrate how comparing designs across design families allows a reasoned trade-off of design structure against estimation precision.

Consider the following questions an analyst might ask concerning the alternative-specific choice design just presented.

1. How much does efficiency increase if 54 choice sets are used instead of two replications of 27 choice sets?
2. What is the efficiency loss if each of the brands (Coke, Pepsi, RC) must be present in a choice set?
3. What is the gain in efficiency if a fourth alternative is added to each choice set?
4. What happens to efficiency if this fourth alternative is constant (e.g., “keep on shopping”)?

The first question assesses the benefit of building a design with 54 choice sets rather than using the original 27 choice sets twice. As Table 4 shows, specifying twice as many choice sets produces a *D*-error of .079 compared with .084 (=.167/2) for two independent runs of the 27 choice set design. This relatively small 6% benefit in efficiency indicates that the original 27 choice set design, while highly fractionated, appears to have suffered little due to this fact.

The second question evaluates the impact of constraints on the choice sets that respondents face. The original design often paired the same brand against itself within a choice set. For example, a choice

[§]The assumption that price has a different impact depending on the brand is testable. The ability to make that test is just one of the advantages of these choice designs.

[¶]We need the fourth two-way interaction, price×container, to be able to estimate the three-way interaction brand×price×container. Of course, there are many other ways of coding a design.

[*]The appendix contains a SAS/IML program that performs the search for this design. Focusing on the principles of the algorithm, the program was deliberately kept simple, specific, and small. A general macro for searching for choice designs, `%ChoicEff`, is documented in Kuhfeld (2010) starting on pages 803 and 806. See page 556 for an example.

Table 4
Impact of Design Modifications on *D*-Error

Design Modification	*D*-error	Efficiency per Choice Set	Comments
27 sets, 3 alternatives per set.	.167	100%	Original design.
Double the number of sets.	.079	106%	Limited benefit from doubling the number of sets.
Require each alternative to contain one of each brand.	.175	95%	Shows minor cost of constraining a design.
Add a fourth alternative.	.144	116%	Diminishing returns from adding additional alternatives.
Fourth alternative is constant.	.195	86%	Design is less efficient because constant alternative is chosen 25% of the time.

set with Coke in a 12 oz bottle for \$5.69 per case might include Coke in a 16 oz bottle for \$7.49 per case. For managerial reasons it might be desirable to have each brand (Coke, Pepsi, RC) represented in every set of three alternatives. To examine the cost of this constraint, Coke is assigned to the first alternative, Pepsi to the second alternative, and RC to the third alternative within each of the 27 choice sets. With this constraint, the *D*-error is .175. This relatively moderate decrease in efficiency of 5% should be acceptable if there are managerially-based reasons to constrain the choice sets.

The third question investigates the benefits of adding a fourth alternative to each choice set. This change increases by 25% the number of alternatives, although the marginal effect of an additional alternative should not be as great. With this modification, *D*-error becomes .144, producing a 16% efficiency gain over three alternatives per choice set. The decision whether to include a fourth alternative now pits the analyst's appraisal of the trade-off between the value of this 16% efficiency gain and the cost in respondent time and reliability.

What happens if this fourth alternative is common and constrained to be constant in all choice sets? With a constant alternative, respondents are not forced to make a choice among undesirable alternatives. Moreover, a constant alternative permits an estimate of demand volume rather than just market shares (Carson et al. 1994). A constant alternative can take many forms, ranging from the respondent's "current brand," to an indication that "none are acceptable," or simply "keep on shopping." While constant alternatives are often added to choice sets, little is known about the efficiency implications of this practice. To create designs with a constant alternative, this alternative must be added to the candidate set. Also, a model with a constant alternative has one more parameter. Comparing a design with a constant alternative to one without, it is necessary to calculate *D*-error with respect to the original 32 parameters using the corresponding submatrix of $\mathbf{\Sigma}$.

Adding a constant alternative to the original design increases the *D*-error of the original 32 parameters by 17% and is nearly 35% worse than allowing the fourth alternative to be variable. Some part of this loss in efficiency is due to the one additional degree of freedom from the constant alternative. A larger part is due to the efficiency lost when respondents are assumed to select the constant alternative. Every time it is chosen, one obtains less information about the values of the other parameters. In this case, the assumption that $\boldsymbol{\beta} = \mathbf{0}$ is not benign, as it assumes the constant alternative, along with all others in the four-option choice sets, will be chosen 25% of the time. We can reduce the efficiency cost to the other parameters by using a smaller $\boldsymbol{\beta}$ for the constant alternative, reflecting the assumption that it will be chosen less often.

In summary, the analysis suggests that adding a constant alternative to a three-alternative choice set can degrade the precision of estimates around the original parameters. Two caveats are important. First, this result will not always occur. We have found some highly fractionated designs where a constant alternative adds to the resolution of the original design. Second, there are studies where a major goal is the estimation of the constant alternative; in that case "oversampling" the constant ensures that its coefficient will be known with greater precision.

An important lesson across these four examples is that one cannot rely on heuristics to guide design strategies, ignoring statistical efficiency. It is generally necessary to test specific design strategies, given anticipated model parameters, to find a good choice design.

Evaluating Model Modifications. The proposed approach can be used to assess modifications of the model specification. This allows one, for example, to estimate the cost of "assumption insurance," i.e., building a design that is robust to false assumptions. Often we assume that factors are independent; for example, that the utility of price does not depend on brand or container. In many instances this assumption would be better termed a "presumption" in that if it is wrong, the estimates are biased, but there is no way to know given the design. Assessing the cost of assumption insurance involves four steps:

1. Find the best design for the unrestricted model (possibly including interactions).
2. Find the best design for the restricted model.
3. Evaluate *D*-error for that unrestricted design under the restricted model.
4. Evaluate *D*-error for the best design for the restricted model.

The cost of assumption insurance is the percent difference between steps 3 and 4, reflecting the loss of efficiency of the core parameters for the two designs. We illustrate how to assess this cost for a design that permits the price term to interact with brand and container versus one that assumes they are independent. To simplify the example, we take the same case as before, but assume that price is a linear rather than a categorical variable.[†]

The first step involves finding an efficient design with all price interactions with brand and brand×container estimable. This unrestricted model has 7 *df* for main effects (two for brand, two for container, two for flavor, and one for price), 12 *df* for two-way interactions (brand×price, brand×flavor, brand×container, container×price), and 4 *df* for the three-way interaction (brand×container×price). An efficient design for this unrestricted model has a *D*-error of .148. If this design is used for a restricted model in which price does not interact (7 *df* for main effects and 8 *df* for two-way interactions) then *D*-error drops

[†]Substituting a linear price term for a three-level categorical one has two immediate implications. First, any change in coding results in quite different absolute values of *D*-error. Second, in optimizing a linear coding for price, the search routine will try to eliminate alternatives with the middle level of price within brand. This focus on extremes is appropriate given the linear assumption, but, may preclude estimation of quadratic effects.

to .118. The critical question is how much better still can one do by searching for the best design in the 15-parameter restricted model. The best design we find has a *D*-error of .110. Thus, assumption insurance in this case imposes a 6% (1 - .110/.118) efficiency loss, a reasonable cost given that prices will often interact with brands and containers.

To summarize, the search routine allows estimates of the cost in efficiency of various design modifications and even changes in the model specification. Again, the important lesson is not the generalizations from the results of these particular examples, but rather an understanding of how these and similar questions can be answered in the context of any research study.

How Good Are These Designs? The preceding discussion has shown that our adaptation of the modified Fedorov algorithm can find estimable choice designs and answer a variety of useful questions. We still need to discuss the question, how close to optimal are these designs? The search is nonexhaustive, and there is no guarantee that the solutions are optimal or even nearly so. For some designs, such as the alternative-specific one shown previously, we can never be completely certain that the search process does not consistently find poor local optima. However, one can achieve some confidence from the pattern of results based on different random restarts; similar efficiencies emerging from different random starts indicate robustness of the resultant designs. An even stronger test is to assess efficiencies of the search process in cases where an optimal solution is known. While this cannot be done generally, we can test the absolute efficiency of certain symmetric designs, where the optimal design can be built using formal methods. We illustrated this kind of design in the three attribute, three level, three alternative, nine choice set ($3^3/3/9$) design discussed earlier, and found that the search routine was not able to find a better design. Now, we ask how good are our generated designs relative to three optimal designs: the design mentioned previously and two corresponding, but bigger designs, $4^4/4/16$ and a $5^5/5/25$ generic design.

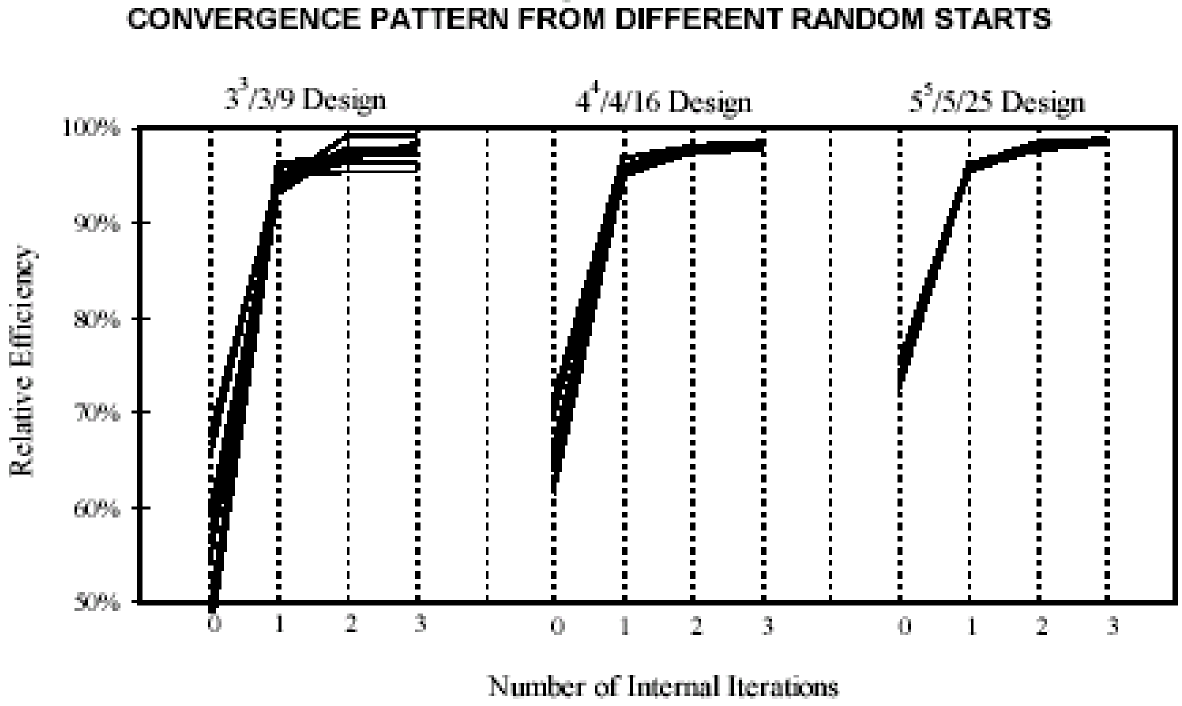
For these types of designs we apply the proposed algorithm and compare our designs with the analytically generated ones. For each design, we used ten different random starts and three internal iterations. Figure 2 displays the impact of efficiency on different starting points and different numbers of internal iterations.

Figure 2 reveals important properties of the proposed algorithm. After the first iteration, the algorithm finds a choice design with about 90% relative efficiency, after a few more iterations, relative efficiencies approach 95%-99%. Further, this property appears to be independent of any initial starting design—the process converges just as quickly from a random start as from a rational one. These encouraging properties suggest important advantages for the practical use of the approach. First, in contrast to Huber and Zwerina (1996), the process does not require a rational starting design (which may be difficult to build). Second, since the process yields very efficient designs after only one or two iterations, most practical problems involving even large choice designs can be accommodated.

Conclusions

We propose an adaptation of the modified Fedorov algorithm to construct statistically efficient choice designs with standard personal computers. The algorithm generates efficient designs quickly and is appropriate for all but the largest choice designs. The approach is illustrated with a SAS/IML program. SAS has the advantage of a general model statement that facilitates the building of choice designs with different model specifications. The cost of using SAS/IML software, however, is that the algorithm generally runs slower than a program developed in, for example, PASCAL or C.

Figure 2
Convergence Pattern From Different Random Starts



There are three major advantages of using a computer to construct choice designs rather than deriving them from formal principles. First, computers are the only way we know to build designs that allow one to incorporate anticipated model parameters. Since the incorporation of this information can increase efficiency by 10% to 50% (see Huber and Zwerina 1996), this benefit alone justifies the use of computer search routines to find efficient choice designs.

The second advantage is that one is less restricted in design selection. Symmetric designs may not reflect the typically asymmetric characteristics of the real market. The adaptability of computerized searches is particularly important in choice studies that simulate consumer choice in a real market (Carson et al. 1994). Moreover, the process we propose allows the analyst to generate choice designs that account for any set of interactions, or alternative-specific effects of interest and critical tests of these assumptions. We illustrated a market emulation design that permits brand to interact with price, container, and flavor and can test the three-way interaction of brand by container by price. This pattern of alternative-specific effects would be very hard to build with standard designs, but it is easy to do with the computerized search routine by simply setting the model statement. The process can handle even more complex models, such as availability and attribute cross-effects models (Lazari and Anderson 1994).

Finally, the ability to assess expected errors in parameters permits the researcher to examine the impact of different modifications in a given design, such as adding more choice sets or dropping a level from a factor. Most valuable, perhaps, is the ability to easily test designs for robustness. We provide one example of assumption insurance, but others are straightforward to generate. What happens to the efficiency of a design if there are interactions, but they are not included in the model statement? What kind of model will do a good job given a linear representation of price, but will also permit a test of curvature? What happens to the efficiency of the design if one's estimate of $\boldsymbol{\beta}$ is wrong?

There are several areas in which future research is needed. The first of these involves studies of the search process per se. We chose the modified Fedorov algorithm because it is robust and runs fast enough on today's desktop computers. As computing power increases, more exhaustive searches should be evaluated. For extremely large problems, faster and less reliable algorithms may be appropriate. Furthermore, while the approach builds efficient choice designs for multinomial logit models, efficiency issues with respect to other models, for example, nested logit and probit models, have yet to be explored.

A second area in which research would be fruitful involves the behavioral impact of different choice designs. The evaluations of our designs all implicitly assumed that the error level is constant regardless of the design. Many choice experiments use relatively small set sizes and few attributes reflecting an implicit recognition that "better" information comes from making the choice less complex. However, from a statistical perspective it is easy to show that smaller set sizes reduce statistical efficiency. In one example, we demonstrated that increasing the number of alternatives per choice set from three to four can increase efficiency by 16%. This gain depends on the assumption that respondent's error levels do not change. If they do increase, then that 16% percent gain might be lessened or even reversed. Thus, there is a need for a series of studies measuring respondents' error levels to tasks at different levels of complexity. Also, it is important to measure the degree of correspondence between the experimental tasks and the actual market behavior, choice experiments are intended to simulate. Such information is critical for correct trade-offs between design efficiency, measured here, and survey effectiveness, measured in the marketplace.

The purpose of this article is to demonstrate the important advantages of a flexible computerized search in generating efficient choice designs. The proposed adaptation of the modified Fedorov algorithm solves many of the practical problems involved in building choice designs, thus enabling more researchers to conduct choice experiments. Nevertheless, we want to emphasize that it does not preclude traditional design skills; they remain critical in determining the model specification and in assessing the choice designs produced by the computerized search.

Appendix

SAS/IML Code for the Proposed Choice Design Algorithm

The SAS code shows a simple implementation of the algorithm. In this example, the program finds a design with 27 choice sets and three alternatives per set. There are four attributes (brand, price, container, and flavor) each with three levels. A design is requested in which all main effects, the two-way interactions between brand and the other attributes, the two-way interaction between container and price, and the brand by price by container three-way interaction are estimable. Here, the parameters are assumed to be zero, but could be easily changed by setting other values.

A computer that evaluated all possible ($81^{81}/3!27! = 5 \times 9 \times 10^{125}$) designs would take numerous billion years. Instead, we use the modified Fedorov algorithm, which uses the following heuristic: find the best exchange for each design point given all of the other candidate points. With 81 candidate alternatives, 27 choice sets, 3 alternatives per set, (say) 3 internal iterations, and 2 random starts, $81 \times 27 \times 3 \times 3 \times 2 = 39,366$ exchanges must be evaluated. The algorithm tries to maximize $|\mathbf{X}'\mathbf{X}|$ rather than minimizing $|(\mathbf{X}'\mathbf{X})^{-1}|$ (note that $|(\mathbf{X}'\mathbf{X})^{-1}| = |\mathbf{X}'\mathbf{X}|^{-1}$). Each exchange requires then the evaluation of a matrix determinant, $|\mathbf{X}'\mathbf{X}|$. Fortunately, we do not have to evaluate this determinant from scratch for each exchange since $|\mathbf{X}'\mathbf{X} + \mathbf{x}'\mathbf{x}| = |\mathbf{X}'\mathbf{X}||I + \mathbf{x}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{x}'|$ (Mardia et al. 1979). Each exchange evaluates a quadratic form, and in this example with three alternatives per choice set, the determinant of a 3×3 matrix. It should also be noted that this algorithm can handle a rank-deficient covariance matrix by operating on $|\mathbf{X}'\mathbf{X} + I\epsilon|$, where ϵ is a small number. This eliminates zero determinants so that less-than-full-rank codings and singular starting designs are not a problem. With these short cuts, one iteration required about 30 seconds on an ordinary 486 PC, implying that the algorithm is reasonable for many marketing contexts.

This appendix is provided simply to show the algorithm for those who might wish to implement or better understand it. If you want to use the algorithm, use the `%ChoicEff` autocall SAS macro documented in Kuhfeld (2010) starting on pages 803 and 806. See page 556 for an example. The `%ChoicEff` is much larger and more full-featured than the code shown in this appendix.

```
/*-------------------------------Initial Set Up-------------------------------*/
%let beta  = 0 0 0 0 0 0 0 0             /* 8 main effects                    */
             0 0 0 0 0 0 0 0             /* brand x price, brand x container,*/
             0 0 0 0                     /* brand x flavor,                   */
             0 0 0 0                     /* price x container interactions   */
             0 0 0 0 0 0 0 0;            /* brand x price x container        */
%let nalts = 3;                          /* Number of alternatives           */
%let nsets = 27;                         /* Number of choice sets            */

proc plan ordered;                       /* Create candidate alternatives    */
   factors brand=3 price=3 contain=3 flavor=3 / noprint;
   output out=candidat;
   run;

proc transreg design data=candidat;      /* Code the candidate alternatives  */
   model class(brand price contain flavor brand*price brand*contain
               brand*flavor contain*price brand*contain*price / effects);
   output out=tmp_cand;
   run;

proc contents p data=tmp_cand(keep=&_trgind); run;

/*------------------------Begin Efficient Design Search-----------------------*/
proc iml; file log;
   use tmp_cand(keep=&_trgind);          /* Identify candidate set for input */
   read all into cand;                   /* Read candidate set into IML      */
   utils  = exp(cand * {&beta}`);        /* exp(alternative utilities)       */
   np     = 1 / ncol(cand);              /* Exponent applied to determinant  */
   imat   = i(&nalts);                   /* Identity matrix                  */
   nobs   = &nsets # &nalts;             /* Total n of alts in choice design */
   ncands = nrow(cand);                  /* Number of candidates             */
   fuzz   = i(ncol(cand)) # 1e-8;        /* X`X ridge factor, avoid singular */

   start center(x, exputil);             /* Probability centering subroutine */
      do i = 1 to nrow(x) / &nalts;      /* Do for each choice set           */
         k = (i-1)#&nalts+1 : i#&nalts;  /* Choice set index vector          */
         p = exputil[k,]; p = p / sum(p); /* Probability of choice           */
         z = x[k,];                      /* Get choice set                   */
         x[k,] = (z - j(&nalts,1,1) *    /* Center choice set, absorb p's    */
                  p` * z) # sqrt(p);
         end;
      finish;

   /*---------------Create Designs With Different Random Starts---------------*/
   do desnum = 1 to 2;                   /* Number of designs to create      */

      indvec  = ceil(ncands *            /* Random index vector (indvec)     */
                uniform(j(1, nobs, 0))); /* into candidates                  */
      des     = cand[indvec,];           /* Initial random design            */
```

```
run center(des, utils[indvec,]);    /* Probability center              */
currdet = det(des` * des);          /* Initial determinants, eff's     */
maxdet  = currdet; oldeff = currdet ## np; fineff = oldeff;
if fineff <= 0 then err = .; else err = 1 / fineff;
put /// +8 'Design   Iteration  D-Efficiency       D-Error' /
        +8 '-----------------------------------------------';
put +6 desnum 6. 0 10. +6 fineff best12. +2 err best12.;

/*-------------------------Internal Iterations-------------------------*/
do iter = 1 to 8 until(converge);   /* Iterate until convergence       */

   /*---------Consider Replacing Each Alternative in the Design---------*/
   do desi = 1 to nobs;             /* Process each alt in design      */
      ind = ceil(desi / &nalts);    /* Choice set number               */
      ind = (ind - 1) # &nalts + 1  /* Choice set index vector         */
            : ind # &nalts;
      besttry = des[ind,];          /* Store current choice set        */
      des[ind,] = 0;                /* Remove current choice set       */
      do i = 0 to 100 until(d ## np > 1e-8);
         xpx = des`*des + i#i*fuzz; /* X`X, ridged if necessary        */
         d   = det(xpx);            /* Determinant, if 0 then X`X will */
         end;                       /* be ridged to make it nonsingular */
      xpxinv = inv(xpx);            /* Inverse (all but current set)   */
      indcan = indvec[,ind];        /* Indvec for this choice set      */
      alt = mod(desi-1, &nalts) + 1;/* Alternative number              */

      /*-----------------Loop Over All of the Candidates----------------*/
      do candi = 1 to ncands;       /* Consider each candidate         */
         indcan[,alt] = candi;      /* Update indvec for this candidate */
         tryit = cand[indcan,];     /* Candidate choice set            */
         run center(tryit,          /* Probability center              */
                    utils[indcan,]);
         currdet = d *              /* Update determinant              */
                   det(imat + tryit * xpxinv * tryit`);

         /*-----------Store Results When Efficiency Improves-----------*/
         if currdet > maxdet then do;
            maxdet       = currdet;/* Best determinant so far         */
            indvec[,desi] = candi;  /* Indvec of best design so far    */
            besttry      = tryit;  /* Best choice set so far          */
            end;
         end;

      des[ind,] = besttry;          /* Update design with new choice set*/
      end;

   /*----------Evaluate Efficiency/Convergence, Report Results----------*/
   newveff = maxdet ## np;          /* Newest efficiency               */
   converge = ((neweff - oldeff) /  /* Less than 1/2 percent           */
            max(oldeff,1e-8) < 0.005); /* improvement means convergence    */
         oldeff = neweff;              /* Store for use in next iteration  */
         fineff = det(des` * des) ## np;  /* Efficiency at end of iteration   */
         if fineff <= 0 then err = .; else err = 1 / fineff;
         put +12 iter 10. +6 fineff best12. +2 err best12.;
         end;

      /*----Store Efficiency, Index of Efficient Design, Covariance Matrix----*/
      final = final // (shape(desnum || fineff, nobs, 2) || indvec`);
      cov = cov // (shape(desnum || fineff, ncol(des), 2) ||
                   sweep(des` * des, 1 : ncol(des)));
      end;

   /*---------------------Write Results to SAS Data Sets---------------------*/
   create cov     var({design effic &_trgind});   append from cov;
   create results var({design effic index   });   append from final;
   quit;

/*-------------Store Actual Design Points, Using Indices from IML-------------*/
data results; set results; i=index; n=_n_; set candidat point=i; run;
proc sort; by descending effic n; run;    /* Put most eff design first       */
proc print; run;                          /* Print designs, best to worst     */
```

Discrete Choice

Warren F. Kuhfeld

Abstract

Discrete choice modeling is a popular technique in marketing research, transportation, and other areas. It is used to help researchers understand people's stated choice of alternative products and services. We discuss designing a choice experiment, preparing the questionnaire, inputting and processing the data, performing the analysis, and interpreting the results.[*] Most of the discussion is on designing the choice experiment.

Introduction

This chapter shows you how to use the multinomial logit model (McFadden 1974; Manski and McFadden 1981; Louviere and Woodworth 1983) to investigate consumer's stated choices. The multinomial logit model is an alternative to full-profile conjoint analysis that is extremely popular in marketing research (Louviere 1991; Carson et. al. 1994). Discrete choice analysis, using the multinomial logit model, is sometimes referred to as "choice-based conjoint." However, the discrete choice model is different from a full-profile conjoint model. Discrete choice analysis uses a nonlinear model and aggregate choice data, whereas full-profile conjoint analysis uses a linear model and individual-level rating or ranking data.

The design and analysis of a discrete choice experiment is explained in the context of a series examples.[†] There are also several very basic introductory examples starting on page 127 in the introduction to experimental design chapter, which starts on page 53. Be sure to read the design chapter before proceeding to the examples in this chapter. The examples are as follows:

- The candy example (page 289) is a first, very simple example that discusses the multinomial logit model, the input data, analysis, results, and computing the probability of choice.
- The fabric softener example (page 302) is a small, somewhat more realistic example that discusses designing the choice experiment, randomization, generating the questionnaire, entering and processing the data, analysis, results, probability of choice, and custom questionnaires.

[*]Copies of this chapter (MR-2010F), the other chapters, sample code, and all of the macros are available on the Web `http://support.sas.com/resources/papers/tnote/tnote_marketresearch.html`. Specifically, sample code is here `http://support.sas.com/techsup/technote/mr2010f.sas`. For help, please contact SAS Technical Support. See page 25 for more information.This document would not be possible without the help of Randy Tobias who contributed to the discussion of experimental design and Ying So who contributed to the discussion of analysis. Randy Tobias wrote PROC FACTEX and PROC OPTEX. Ying So wrote PROC PHREG. Warren F. Kuhfeld wrote PROC TRANSREG and all of the macros.

[†]All of the example data sets are artificially generated.

- The first vacation example (page 339) is a larger, symmetric example (all factors have the same number of levels) that discusses designing the choice experiment, blocks, randomization, generating the questionnaire, entering and processing the data, coding, and alternative-specific effects.
- The second vacation example (page 410) is a larger, asymmetric example (not all factors have the same number of levels) that discusses designing the choice experiment, blocks, blocking an existing design, interactions, generating the questionnaire, generating artificial data, reading, processing, and analyzing the data, aggregating the data to save time and memory.
- The brand choice example (page 444) is a small example that discusses the processing of aggregate data, the mother logit model, and the likelihood function.
- The food product example (page 468) is a medium sized example that discusses asymmetry, coding, checking the design to ensure that all effects are estimable, price cross-effects, availability cross-effects, interactions, overnight design searches, modeling subject attributes, and designs when balance is of primary importance.
- The drug allocation example (page 535) is a small example that discusses data processing for studies where respondents potentially make multiple choices.
- The chair example (page 556) is a purely generic-attributes study, and it uses the `%ChoicEff` macro to create experimental designs.
- The next example section (page 580) shows how to improve an existing design and augmenting a design with some choice sets are fixed in advance.
- The last example section (page 595) discusses partial-profile designs and designs with restrictions. Also see page 1079 for an example of a choice design with a complicated set of restrictions.

This chapter relies heavily on a number of macros and procedures.

- The `%MktRuns` autocall macro suggests design sizes. See page 1159 for documentation.
- The `%MktEx` autocall macro generates designs for linear models. These designs are directly used for conjoint studies. After post-processing they can also be used as choice designs. They are also used to make candidate sets for directly constructing choice designs. Most examples use the `%MktEx` macro in some capacity. See page 1017 for documentation.
- The `%MktEval` autocall macro evaluates linear model designs. See page 1012 for documentation.
- The `%ChoicEff` autocall macro both generates and evaluates choice designs. See page 806 for documentation.
- The autocall macros `%MktKey`, `%MktRoll`, `%MktMerge`, and `%MktAllo` prepare the data and design for analysis. See pages 1090, 1153, 1125, and 956 for documentation.
- PROC TRANSREG codes our designs, which puts the data into the final form for analysis.
- The `%PHChoice` autocall macro customizes our displayed output. This macro uses PROC TEMPLATE and ODS (Output Delivery System) to customize the output from PROC PHREG, which fits the multinomial logit model. See page 1173 for documentation.
- The `%MktBal` macro makes perfectly balanced designs for main effects models. See page 959 for documentation.

- The `%MktBlock` macro block a linear or choice design into sets of alternatives or choice sets. See page 979 for documentation.
- The `%MktDups` macro searches a design for duplicate runs or choice sets. See page 1004 for documentation.
- The `%MktLab` macro assigns variable names, labels and levels to experimental designs and adds an intercept. See page 1093 for documentation.
- The `%MktOrth` macro lists the orthogonal experimental designs that the `%MktEx` macro can produce. See page 1128 for documentation.
- The `%MktPPro` macro makes certain partial-profile choice designs. See page 1145 for documentation.
- The `%MktBIBD` macro makes balanced incomplete block designs, unbalanced block designs, and more generally, incomplete block designs, which are useful in constructing certain partial-profile designs and MaxDiff designs. See page 963 for documentation.
- The `%MktMDiff` macro processes and analyzes data from MaxDiff (best-worst) studies. See page 1105 for documentation.

All of these macros are distributed with SAS 9.2 as autocall macros (see page 803 for more information about autocall macros), however, you should get the latest versions of the macros from the Web `http://support.sas.com/resources/papers/tnote/tnote_marketresearch.html`.

Experimental Design

Experimental design is a fundamental component of choice modeling. A discrete choice experimental design consists of sets of products, and subjects choose a product from each set. Often, the most challenging part of the entire study is making the design. There are many examples of making choice designs in this chapter. Before you read them, be sure to read the design chapter beginning on page 53. There are also a number of design examples with the macro documentation. After you become familiar with the design chapter, you should check out the `%ChoicEff` macro examples starting on page 808.

Customizing the Multinomial Logit Output

You can fit the multinomial logit model for discrete choice experiments by using the SAS/STAT procedure PHREG (proportional hazards regression), with the `ties=breslow` option. The likelihood function for the multinomial logit model has the same form as a survival analysis model fit by PROC PHREG. PROC PHREG and its output are primarily designed for survival-analysis studies. Before you fit the multinomial logit model with PROC PHREG, you can customize its output templates to make them more appropriate for choice experiments by using the `%PHChoice` autocall macro. See page 803 for information about autocall macros. You can run the following macro to customize PROC PHREG output:

```
%phchoice(on)
```

The macro uses PROC TEMPLATE and ODS (Output Delivery System) to customize the output from PROC PHREG. Running this step edits two of the PROC PHREG templates and stores copies in the `sasuser` library. The default templates that SAS provides are stored in the library `sashelp.tmplmst`. By default, if you modify a template, it is stored in the library `sasuser.templat`. By default, ODS searches `sasuser.templat` for templates, and then it searches `sashelp.tmplmst` if it does not find the requested template in `sasuser.templat`. You can see the list of template libraries by submitting the following statement:

```
ods path show;
```

The template changes made by the `%PHChoice` macro do not affect the numerical output from PROC PHREG, but they do change the format and some of the titles, labels, and column headers that are used to label the output. Note that these changes assume that each effect in the choice model has a variable label associated with it, so there is no need to display variable names. If you are coding with PROC TRANSREG, this is usually the case. These changes remain in effect until you delete them. To return to the default output from PROC PHREG, run the following macro:

```
%phchoice(off)
```

More generally, you can run the following step to delete the entire `sasuser.templat` library of customized templates so that ODS uses only the SAS supplied templates:

```
proc datasets library=sasuser;
   delete templat(memtype=itemstor);
   run;
```

If you only use PROC PHREG for choice modeling, and if you do not delete the contents of the `sasuser` or the template library, you only have to run the `%PHChoice` macro once, and the template changes are there for all subsequent steps that use the same template library in the `sasuser` library. Alternatively, you can choose to run `phchoice(on)` before running PROC PHREG and `phchoice(off)` after you are done. If you plan on using PROC PHREG for survival analysis, you should be sure to run `phchoice(off)` first so that your output is labeled appropriately for a survival study. See page 1173 for more information about the `%PHChoice` macro.

Candy Example

We begin with a very simple introductory example. In this example, we discuss the multinomial logit model, data input and processing, analysis, results, interpretation, and probability of choice. Many aspects of this example, the experimental design in particular, are simpler than almost all realistic choice studies. Still, it is useful to start with a simple choice study with no experimental design issues to consider. In this example, each of ten subjects is presented with eight different chocolate candies and asked to choose one. The eight candies consist of the 2^3 combinations of dark or milk chocolate, soft or chewy center, and nuts or no nuts. Each subject saw all eight candies and made one choice. Experimental choice data such as these are typically analyzed with a multinomial logit model.

The Multinomial Logit Model

The multinomial logit model assumes that the probability that an individual will choose one of the m alternatives, c_i, from choice set C is

$$p(c_i|C) = \frac{\exp(U(c_i))}{\sum_{j=1}^{m} \exp(U(c_j))} = \frac{\exp(\mathbf{x}_i \boldsymbol{\beta})}{\sum_{j=1}^{m} \exp(\mathbf{x}_j \boldsymbol{\beta})}$$

where $\mathbf{x}_i$ is a vector of coded attributes and $\boldsymbol{\beta}$ is a vector of unknown attribute parameters. At the heart- of this formula is a linear part-worth utility function, $\mathbf{x}_i\boldsymbol{\beta}$ (like a linear regression function, but wrapped in a more complicated nonlinear model). $U(c_i) = \mathbf{x}_i\boldsymbol{\beta}$ is the utility for alternative c_i, which is a linear function of the attributes. The probability that an individual will choose one of the m alternatives, c_i, from choice set C is the exponential of the utility of the alternative divided by the sum of all of the exponentiated utilities.

There are $m = 8$ attribute vectors in this example, one for each alternative. Let $\mathbf{x}$ = (Dark/Milk, Soft/Chewy, Nuts/No Nuts) where Dark/Milk = (1 = Dark, 0 = Milk), Soft/Chewy = (1 = Soft, 0 = Chewy), Nuts/No Nuts = (1 = Nuts, 0 = No Nuts). The eight attribute vectors are

$\mathbf{x}_1 = (0\ 0\ 0)$	(Milk, Chewy, No Nuts)
$\mathbf{x}_2 = (0\ 0\ 1)$	(Milk, Chewy, Nuts)
$\mathbf{x}_3 = (0\ 1\ 0)$	(Milk, Soft, No Nuts)
$\mathbf{x}_4 = (0\ 1\ 1)$	(Milk, Soft, Nuts)
$\mathbf{x}_5 = (1\ 0\ 0)$	(Dark, Chewy, No Nuts)
$\mathbf{x}_6 = (1\ 0\ 1)$	(Dark, Chewy, Nuts)
$\mathbf{x}_7 = (1\ 1\ 0)$	(Dark, Soft, No Nuts)
$\mathbf{x}_8 = (1\ 1\ 1)$	(Dark, Soft, Nuts)

Say, hypothetically that $\boldsymbol{\beta}' = (4 \quad -2 \quad 1)$. That is, the part-worth utility for dark chocolate is 4, the part-worth utility for soft center is -2, and the part-worth utility for nuts is 1. The utility for each of the combinations, $\mathbf{x}_i\boldsymbol{\beta}$, is as follows:

U(Milk, Chewy, No Nuts)	=	0×4	+	0×-2	+	0×1	=	0
U(Milk, Chewy, Nuts)	=	0×4	+	0×-2	+	1×1	=	1
U(Milk, Soft, No Nuts)	=	0×4	+	1×-2	+	0×1	=	-2
U(Milk, Soft, Nuts)	=	0×4	+	1×-2	+	1×1	=	-1
U(Dark, Chewy, No Nuts)	=	1×4	+	0×-2	+	0×1	=	4
U(Dark, Chewy, Nuts)	=	1×4	+	0×-2	+	1×1	=	5
U(Dark, Soft, No Nuts)	=	1×4	+	1×-2	+	0×1	=	2
U(Dark, Soft, Nuts)	=	1×4	+	1×-2	+	1×1	=	3

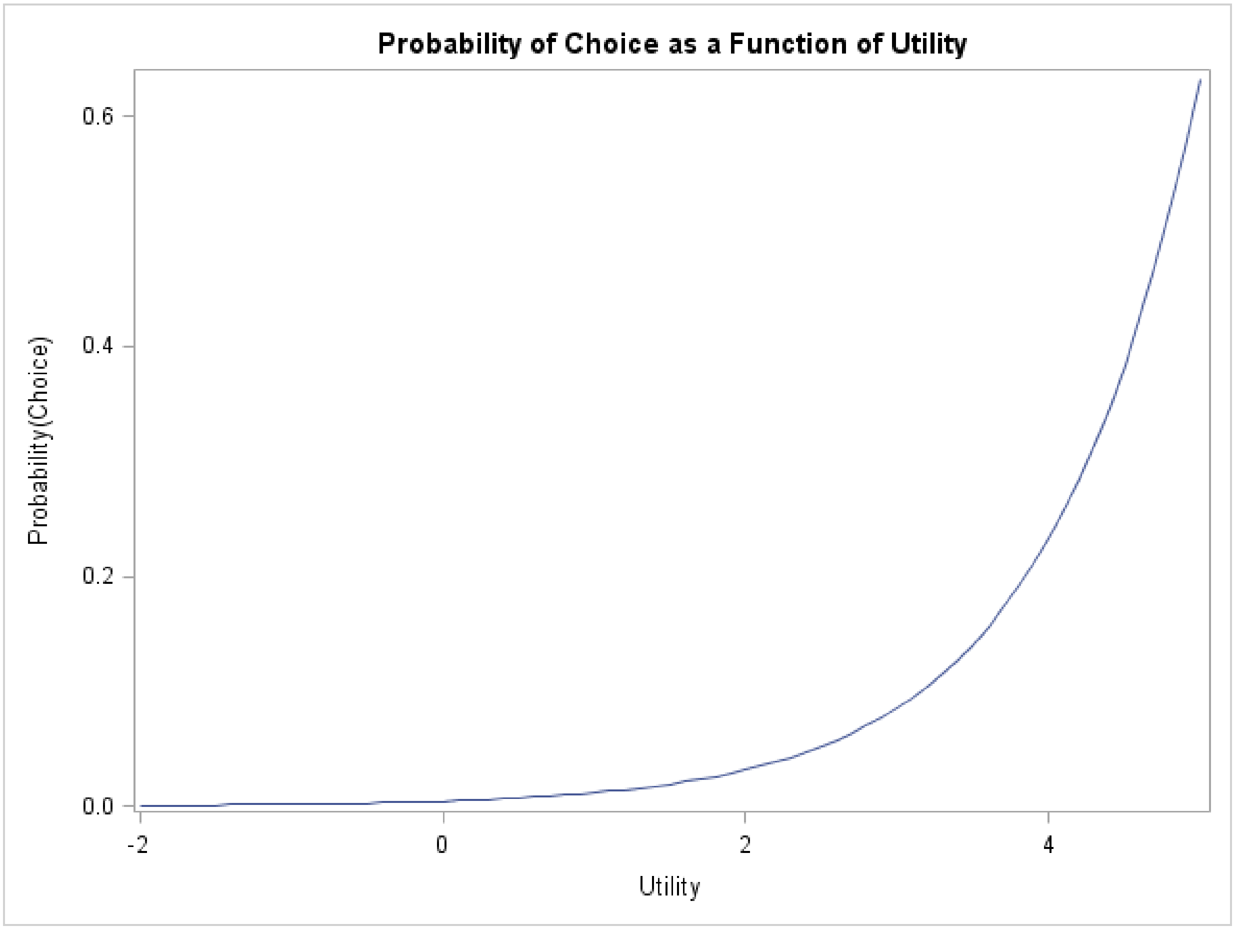
The denominator of the probability formula, $\sum_{j=1}^{m} \exp(\mathbf{x}_j \boldsymbol{\beta})$, is $\exp(0) + \exp(1) + \exp(-2) + \exp(-1) + \exp(4) + \exp(5) + \exp(2) + \exp(3) = 234.707$. The probability that each alternative is chosen, $\exp(\mathbf{x}_i \boldsymbol{\beta}) / \sum_{j=1}^{m} \exp(\mathbf{x}_j \boldsymbol{\beta})$, is

p(Milk, Chewy, No Nuts)	=	exp(0)	/	234.707	=	0.004
p(Milk, Chewy, Nuts)	=	exp(1)	/	234.707	=	0.012
p(Milk, Soft, No Nuts)	=	exp(-2)	/	234.707	=	0.001
p(Milk, Soft, Nuts)	=	exp(-1)	/	234.707	=	0.002
p(Dark, Chewy, No Nuts)	=	exp(4)	/	234.707	=	0.233
p(Dark, Chewy, Nuts)	=	exp(5)	/	234.707	=	0.632
p(Dark, Soft, No Nuts)	=	exp(2)	/	234.707	=	0.031
p(Dark, Soft, Nuts)	=	exp(3)	/	234.707	=	0.086

Note that even combinations with a negative or zero utility have a nonzero probability of choice. Also note that adding a constant to the utilities does not change the probability of choice, however multiplying by a constant will.

Probability of choice is a nonlinear and increasing function of utility. The plot produced by the following steps shows the relationship between utility and probability of choice for this hypothetical situation:

```
data x;
   do u = -2 to 5 by 0.1;
      p = exp(u) / 234.707;
      output;
      end;
   label p = 'Probability(Choice)' u = 'Utility';
   run;

proc sgplot data=x;
   title 'Probability of Choice as a Function of Utility';
   series y=p x=u;
   run;
```



This plot shows the function $\exp(-2)$ to $\exp(5)$, scaled into the range zero to one, the range of probability values. For the small negative utilities, the probability of choice is essentially zero. As utility increases beyond two, the function starts rapidly increasing.

In this example, the chosen alternatives are $\mathbf{x}_5$, $\mathbf{x}_6$, $\mathbf{x}_7$, $\mathbf{x}_5$, $\mathbf{x}_2$, $\mathbf{x}_6$, $\mathbf{x}_2$, $\mathbf{x}_6$, $\mathbf{x}_6$, $\mathbf{x}_6$. Alternative $\mathbf{x}_2$ is chosen 2 times, $\mathbf{x}_5$ is chosen 2 times, $\mathbf{x}_6$ is chosen 5 times, and $\mathbf{x}_7$ is chosen 1 time. The choice model likelihood for these data is the product of ten terms, one for each choice set for each subject. Each term consists of the probability that the chosen alternative is chosen. For each choice set, the utilities for all of the alternatives enter into the denominator, and the utility for the chosen alternative enters into the numerator. The choice model likelihood for these data is

$$\begin{aligned}
\mathcal{L}_C &= \frac{\exp(\mathbf{x}_5\boldsymbol{\beta})}{\left[\sum_{j=1}^{8}\exp(\mathbf{x}_j\boldsymbol{\beta})\right]} \times \frac{\exp(\mathbf{x}_6\boldsymbol{\beta})}{\left[\sum_{j=1}^{8}\exp(\mathbf{x}_j\boldsymbol{\beta})\right]} \times \frac{\exp(\mathbf{x}_7\boldsymbol{\beta})}{\left[\sum_{j=1}^{8}\exp(\mathbf{x}_j\boldsymbol{\beta})\right]} \times \frac{\exp(\mathbf{x}_5\boldsymbol{\beta})}{\left[\sum_{j=1}^{8}\exp(\mathbf{x}_j\boldsymbol{\beta})\right]} \times \\
&\quad \frac{\exp(\mathbf{x}_2\boldsymbol{\beta})}{\left[\sum_{j=1}^{8}\exp(\mathbf{x}_j\boldsymbol{\beta})\right]} \times \frac{\exp(\mathbf{x}_6\boldsymbol{\beta})}{\left[\sum_{j=1}^{8}\exp(\mathbf{x}_j\boldsymbol{\beta})\right]} \times \frac{\exp(\mathbf{x}_2\boldsymbol{\beta})}{\left[\sum_{j=1}^{8}\exp(\mathbf{x}_j\boldsymbol{\beta})\right]} \times \frac{\exp(\mathbf{x}_6\boldsymbol{\beta})}{\left[\sum_{j=1}^{8}\exp(\mathbf{x}_j\boldsymbol{\beta})\right]} \times \\
&\quad \frac{\exp(\mathbf{x}_6\boldsymbol{\beta})}{\left[\sum_{j=1}^{8}\exp(\mathbf{x}_j\boldsymbol{\beta})\right]} \times \frac{\exp(\mathbf{x}_6\boldsymbol{\beta})}{\left[\sum_{j=1}^{8}\exp(\mathbf{x}_j\boldsymbol{\beta})\right]} \\
&= \frac{\exp((2\mathbf{x}_2 + 2\mathbf{x}_5 + 5\mathbf{x}_6 + \mathbf{x}_7)\boldsymbol{\beta})}{\left[\sum_{j=1}^{8}\exp(\mathbf{x}_j\boldsymbol{\beta})\right]^{10}}
\end{aligned}$$

The Input Data

The data set consists of one observation for each alternative of each choice set for each subject. (A typical choice study has more than one choice set per person. This first example only has one choice set to help keep it simple.) All of the chosen and unchosen alternatives must appear in the data set. The data set must contain variables that identify the subject, the choice set, which alternative is chosen, and the set of alternatives from which it is chosen. In this example, the data set contains $10 \times 1 \times 8 = 80$ observations: 10 subjects each saw 1 choice set with 8 alternatives.

Typically, two or three variables are used to identify the choice sets. These variables include subject ID and choice set number. In addition, larger studies might have a block ID variable when subjects see only a block of choice sets instead of every choice set. In this simple case where each subject only made one choice, the choice set variable is not necessary. However, we use it here to illustrate the more general case. The variable `Subj` is the subject number, and `Set` identifies the choice set within subject. The chosen alternative is indicated by `c=1`, which means first choice. All second and subsequent choices are unobserved, so the unchosen alternatives are indicated by `c=2`, which means that all we know is that they would have been chosen after the first choice (as a second or subsequent choice).

Both the chosen and the unchosen alternatives must appear in the input data set since both are needed to construct the likelihood function. The `c=2` observations enter into the denominator of the likelihood function, and the `c=1` observations enter into both the numerator and the denominator. The following statements read the data and store them in a SAS data set:

```
title 'Choice of Chocolate Candies';

data chocs;
   input Subj c Dark Soft Nuts @@;
   Set = 1;
   datalines;
 1 2 0 0 0    1 2 0 0 1    1 2 0 1 0    1 2 0 1 1
 1 1 1 0 0    1 2 1 0 1    1 2 1 1 0    1 2 1 1 1
 2 2 0 0 0    2 2 0 0 1    2 2 0 1 0    2 2 0 1 1
 2 2 1 0 0    2 1 1 0 1    2 2 1 1 0    2 2 1 1 1
 3 2 0 0 0    3 2 0 0 1    3 2 0 1 0    3 2 0 1 1
 3 2 1 0 0    3 2 1 0 1    3 1 1 1 0    3 2 1 1 1
 4 2 0 0 0    4 2 0 0 1    4 2 0 1 0    4 2 0 1 1
 4 1 1 0 0    4 2 1 0 1    4 2 1 1 0    4 2 1 1 1
 5 2 0 0 0    5 1 0 0 1    5 2 0 1 0    5 2 0 1 1
 5 2 1 0 0    5 2 1 0 1    5 2 1 1 0    5 2 1 1 1
 6 2 0 0 0    6 2 0 0 1    6 2 0 1 0    6 2 0 1 1
 6 2 1 0 0    6 1 1 0 1    6 2 1 1 0    6 2 1 1 1
 7 2 0 0 0    7 1 0 0 1    7 2 0 1 0    7 2 0 1 1
 7 2 1 0 0    7 2 1 0 1    7 2 1 1 0    7 2 1 1 1
 8 2 0 0 0    8 2 0 0 1    8 2 0 1 0    8 2 0 1 1
 8 2 1 0 0    8 1 1 0 1    8 2 1 1 0    8 2 1 1 1
 9 2 0 0 0    9 2 0 0 1    9 2 0 1 0    9 2 0 1 1
 9 2 1 0 0    9 1 1 0 1    9 2 1 1 0    9 2 1 1 1
10 2 0 0 0   10 2 0 0 1   10 2 0 1 0   10 2 0 1 1
10 2 1 0 0   10 1 1 0 1   10 2 1 1 0   10 2 1 1 1
;
```

In this DATA step, the data for four alternatives appear on one line, and all of the data for a choice set of eight alternatives appear on two lines. The DATA step shows the data entry in the way that requires the fewest programming statements. Each execution of the `input` statement reads information about one alternative. The `@@` in the `input` statement specifies that SAS should not automatically go to a new input data set line when it reads the next row of data. This specification is needed here because each line in the input data set contains the data for four output data set rows. The data from the first two subjects is displayed as follows:

```
proc print data=chocs noobs;
   where subj <= 2;
   var subj set c dark soft nuts;
   run;
```

The data for the first two subjects is as follows:

```
                    Choice of Chocolate Candies

            Subj    Set    c    Dark    Soft    Nuts

              1      1     2     0       0       0
              1      1     2     0       0       1
              1      1     2     0       1       0
              1      1     2     0       1       1
              1      1     1     1       0       0
              1      1     2     1       0       1
              1      1     2     1       1       0
              1      1     2     1       1       1
              2      1     2     0       0       0
              2      1     2     0       0       1
              2      1     2     0       1       0
              2      1     2     0       1       1
              2      1     2     1       0       0
              2      1     1     1       0       1
              2      1     2     1       1       0
              2      1     2     1       1       1
```

These next steps illustrate a more typical form of data entry. The experimental design and the data are stored in separate data sets. Then they are merged and processed to produce the same results as the preceding steps. The process of merging the experimental design and the data is explicitly illustrated here with a DATA step program. In practice, and in all of the other examples, we use the `%MktMerge` macro to do this. The following steps read the design, merge it with the data, and designate first and subsequent choices:

```
title 'Choice of Chocolate Candies';

* Alternative Form of Data Entry;

data combos;                     /* Read the design matrix.     */
   input Dark Soft Nuts;
   datalines;
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
;

data chocs;                      /* Create the data set.        */
   input Choice @@; drop choice; /* Read the chosen combo num.  */
   Subj = _n_; Set = 1;          /* Store subj, choice set num. */
   do i = 1 to 8;                /* Loop over alternatives.     */
      c = 2 - (i eq choice);     /* Designate chosen alt.       */
      set combos point=i;        /* Read design matrix.         */
      output;                    /* Output the results.         */
      end;
   datalines;
5 6 7 5 2 6 2 6 6 6
;
```

The variable `Choice` is the number of the chosen alternative. For each choice set, each of the eight observations in the experimental design is read. The `point=` option in the `set` statement is used to read the *ith* observation of the data set `Combos`. When `i` (the alternative index) equals `Choice` (the number of the chosen alternative), the logical expression `(i eq choice)` equals 1; otherwise it is 0. The statement `c = 2 - (i eq choice)` sets `c` to 1 (two minus one) when the alternative is chosen and 2 (two minus zero) otherwise. All eight observations in the `Combos` data set are read 10 times, once per subject. The resulting data set is the same as the one we created previously. As we mentioned previously, in all of the remaining examples, we simplify this process by using the `%MktMerge` macro to merge the design and data and flag the chosen and unchosen alternatives. Still, it is good to know what the `%MktMerge` step does. The basic logic underlying this macro is shown in the preceding step. The number of a chosen alternative is read, then each alternative of the choice set is read, the chosen alternative is flagged `(c = 1)`, and the unchosen alternatives are flagged `(c = 2)`. One observation per choice set per subject is read from the input data, and one observation per alternative per choice set per subject is written.

Choice and Survival Models

In SAS, the multinomial logit model is fit with the SAS/STAT procedure PHREG (proportional hazards regression), with the `ties=breslow` option. The likelihood function of the multinomial logit model has the same form as a survival-analysis model fit by PROC PHREG.

In a discrete choice study, subjects are presented with sets of alternatives and are asked to choose the most preferred alternative. The data for one choice set consist of one alternative that is chosen and $m-1$ alternatives that are not chosen. First choice is observed. Second and subsequent choices are not observed; it is only known that the other alternatives would have been chosen after the first choice. In survival analysis, subjects (rats, people, light bulbs, machines, and so on) are followed until a specific event occurs (such as failure or death) or until the experiment ends. The data are event times. The data for subjects who have not experienced the event (such as those who survive past the end of a medical experiment) are *censored*. The exact event time is not known, but it is known to have occurred after the censored time. In a discrete choice study, first choice occurs at time one, and all subsequent choices (second choice, third choice, and so on) are unobserved or censored. The survival and choice models are the same.

Fitting the Multinomial Logit Model

The preceding steps arranged the data into the right form for analysis. You use PROC PHREG to fit the multinomial logit model as follows:

```
proc phreg data=chocs outest=betas;
   strata subj set;
   model c*c(2) = dark soft nuts / ties=breslow;
   label dark = 'Dark Chocolate' soft = 'Soft Center'
         nuts = 'With Nuts';
   run;
```

The `data=` option specifies the input data set. The `outest=` option requests an output data set called `Betas` with the parameter estimates. The `strata` statement specifies that each combination of the variables `Set` and `Subj` forms a set from which a choice is made. Each term in the likelihood function is a *stratum*. There is one term or stratum per choice set per subject, and each is composed of information about the chosen and all of the unchosen alternatives.

In the left side of the `model` statement, you specify the variables that indicate which alternatives are chosen and not chosen. While this could be two different variables, we use one variable `c` to provide both pieces of information. The response variable `c` has values 1 (chosen or first choice) and 2 (unchosen or subsequent choices). The first `c` of the `c*c(2)` in the `model` statement specifies that `c` indicates which alternative is chosen. The second `c` specifies that `c` indicates which alternatives are not chosen, and `(2)` means that observations with values of 2 are not chosen. When `c` is set up such that 1 indicates the chosen alternative and 2 indicates the unchosen alternatives, always specify `c*c(2)` on the left of the equal sign in the `model` statement. The attribute variables are specified after the equal sign. Specify `ties=breslow` after a slash to explicitly specify the likelihood function for the multinomial logit model. (Do not specify any other `ties=` options; `ties=breslow` specifies the most computationally efficient and always appropriate way to fit the multinomial logit model.) The `label` statement is added since we are using a template that assumes each variable has a label.

Note that the `c*c(`*n*`)` syntax allows second choice `(c=2)` and subsequent choices (`c=3`, `c=4`, ...) to be entered. Just enter in parentheses one plus the number of choices actually made. For example, with first and second choice data specify `c*c(3)`. Note, however, that most experts believe that second and subsequent choice data are much less reliable than first choice data (or last choice data).

Multinomial Logit Model Results

Recall that we specified `%phchoice(on)` on page 287 to customize the output from PROC PHREG. The results are as follows:

```
                         Choice of Chocolate Candies

                             The PHREG Procedure

                              Model Information

                   Data Set                 WORK.CHOCS
                   Dependent Variable       c
                   Censoring Variable       c
                   Censoring Value(s)       2
                   Ties Handling            BRESLOW

                 Number of Observations Read          80
                 Number of Observations Used          80

      Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                                      Number of          Chosen          Not
Stratum    Subj          Set       Alternatives    Alternatives       Chosen

      1    1             1                    8               1            7
      2    2             1                    8               1            7
      3    3             1                    8               1            7
      4    4             1                    8               1            7
      5    5             1                    8               1            7
      6    6             1                    8               1            7
      7    7             1                    8               1            7
      8    8             1                    8               1            7
      9    9             1                    8               1            7
     10    10            1                    8               1            7
---------------------------------------------------------------------------
  Total                                      80              10           70

                              Convergence Status

                Convergence criterion (GCONV=1E-8) satisfied.

                            Model Fit Statistics

                                 Without           With
                 Criterion     Covariates     Covariates

                 -2 LOG L          41.589         28.727
                 AIC               41.589         34.727
                 SBC               41.589         35.635
```

```
                    Testing Global Null Hypothesis: BETA=0

                Test                 Chi-Square       DF     Pr > ChiSq

                Likelihood Ratio        12.8618        3         0.0049
                Score                   11.6000        3         0.0089
                Wald                     8.9275        3         0.0303

                          Choice of Chocolate Candies

                              The PHREG Procedure

                     Multinomial Logit Parameter Estimates

                             Parameter      Standard
                      DF      Estimate         Error    Chi-Square    Pr > ChiSq

    Dark Chocolate     1       1.38629       0.79057        3.0749        0.0795
    Soft Center        1      -2.19722       1.05409        4.3450        0.0371
    With Nuts          1       0.84730       0.69007        1.5076        0.2195
```

The first table, `Model Information`, contains the input data set name, dependent variable name, censoring information, and tie handling option. The next table shows that 80 observations are read from the input SAS data set, and all 80 of them are used in the analysis.

The `Summary of Subjects, Sets, and Chosen and Unchosen Alternatives` table is displayed by default and should be used to check the data entry. There are as many strata as there are combinations of the `Subj` and `Set` variables.[*] In this case, there are ten strata. Each stratum must be composed of m alternatives. In this case, there are eight alternatives. The number of chosen alternatives should be 1, and the number of unchosen alternatives is $m - 1$ (in this case, 7). **Always check the summary table to ensure that the data are arrayed correctly.**

The `Convergence Status` table shows that the iterative algorithm successfully converged. The next tables, `Model Fit Statistics` and `Testing Global Null Hypothesis: BETA=0` contain the overall fit of the model. The –2 LOG L statistic under `With Covariates` is 28.727 and the Chi-Square statistic is 12.8618 with 3 *df* (p=0.0049), which is used to test the null hypothesis that the attributes do not influence choice. At common alpha levels such as 0.05 and 0.01, we reject the null hypothesis of no relationship between choice and the attributes. Note that 41.589 (–2 LOG L Without Covariates, which is –2 LOG L for a model with no explanatory variables) minus 28.727 (–2 LOG L With Covariates, which is –2 LOG L for a model with all explanatory variables) equals 12.8618 (Model Chi-Square, which is used to test the effects of the explanatory variables).

The `Multinomial Logit Parameter Estimates` table is next. For each effect, it contains the maximum likelihood parameter estimate, its estimated standard error (the square root of the corresponding diagonal element of the estimated variance matrix), the Wald Chi-Square statistic (the square of the parameter estimate divided by its standard error), the *df* of the Wald Chi-Square statistic (1 unless the corresponding parameter is redundant or infinite, in which case the value is 0), and the *p*-value of the Chi-Squared statistic with respect to a chi-squared distribution with one *df*. The parameter estimate

[*]More generally, there are as many strata as there are combinations of the `Subj`, `Set`, and block ID variable. In this case, there is only one block and no blocking variable.

with the smallest p-value is for soft center. Since the parameter estimate is negative, chewy is the more preferred level. Dark is preferred over milk, and nuts over no nuts, however only the p-value for Soft is less than 0.05.

Fitting the Multinomial Logit Model, All Levels

It is instructive to perform some manipulations on the data set and analyze it again. These steps perform the same analysis as before, only now, coefficients for both levels of the three attributes are displayed:

```
data chocs2;
   set chocs;
   Milk = 1 - dark; Chewy = 1 - Soft; NoNuts = 1 - nuts;
   label dark = 'Dark Chocolate' milk   = 'Milk Chocolate'
         soft = 'Soft Center'    chewy  = 'Chewy Center'
         nuts = 'With Nuts'      nonuts = 'No Nuts';
   run;

proc phreg data=chocs2;
   strata subj set;
   model c*c(2) = dark milk soft chewy nuts nonuts / ties=breslow;
   run;
```

Binary variables for the missing levels are created by subtracting the existing binary variables from 1. The output is as follows:

```
                         Choice of Chocolate Candies

                             The PHREG Procedure

                              Model Information

                    Data Set                 WORK.CHOCS2
                    Dependent Variable       c
                    Censoring Variable       c
                    Censoring Value(s)       2
                    Ties Handling            BRESLOW

                  Number of Observations Read          80
                  Number of Observations Used          80
```

```
         Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                                     Number of          Chosen         Not
Stratum    Subj        Set        Alternatives    Alternatives      Chosen

      1    1             1                   8               1           7
      2    2             1                   8               1           7
      3    3             1                   8               1           7
      4    4             1                   8               1           7
      5    5             1                   8               1           7
      6    6             1                   8               1           7
      7    7             1                   8               1           7
      8    8             1                   8               1           7
      9    9             1                   8               1           7
     10    10            1                   8               1           7
---------------------------------------------------------------------------
  Total                                     80              10          70

                               Convergence Status

                  Convergence criterion (GCONV=1E-8) satisfied.

                             Model Fit Statistics

                                 Without           With
                   Criterion    Covariates    Covariates

                   -2 LOG L         41.589        28.727
                   AIC              41.589        34.727
                   SBC              41.589        35.635

                   Testing Global Null Hypothesis: BETA=0

           Test                 Chi-Square       DF     Pr > ChiSq

           Likelihood Ratio        12.8618        3         0.0049
           Score                   11.6000        3         0.0089
           Wald                     8.9275        3         0.0303
```

```
                          Choice of Chocolate Candies

                              The PHREG Procedure

                     Multinomial Logit Parameter Estimates

                           Parameter      Standard
                    DF      Estimate         Error    Chi-Square    Pr > ChiSq

   Dark Chocolate    1       1.38629       0.79057        3.0749        0.0795
   Milk Chocolate    0             0             .             .             .
   Soft Center       1      -2.19722       1.05409        4.3450        0.0371
   Chewy Center      0             0             .             .             .
   With Nuts         1       0.84730       0.69007        1.5076        0.2195
   No Nuts           0             0             .             .             .
```

Now, the zero coefficients for the reference levels, milk, chewy, and no nuts are displayed. The part-worth utility for Milk Chocolate is a structural zero, and the part-worth utility for Dark Chocolate is larger at 1.38629. Similarly, the part-worth utility for Chewy Center is a structural zero, and the part-worth utility for Soft Center is smaller at −2.19722. Finally, the part-worth utility for No Nuts is a structural zero, and the part-worth utility for Nuts is larger at 0.84730.

Probability of Choice

The parameter estimates are used next to construct the estimated probability that each alternative is chosen. The DATA step program uses the following formula to create the choice probabilities:

$$p(c_i|C) = \frac{\exp(\mathbf{x}_i\boldsymbol{\beta})}{\sum_{j=1}^{m} \exp(\mathbf{x}_j\boldsymbol{\beta})}$$

```
* Estimate the probability that each alternative is chosen;

data p;
   retain sum 0;
   set combos end=eof;


   * On the first pass through the DATA step (_n_ is the pass
     number), get the regression coefficients in B1-B3.
     Note that they are automatically retained so that they
     can be used in all passes through the DATA step.;

   if _n_ = 1 then
      set betas(rename=(dark=b1 soft=b2 nuts=b3));
   keep dark soft nuts p;
   array x[3] dark soft nuts;
   array b[3] b1-b3;
```

```
      * For each combination, create x * b;
      p = 0;
      do j = 1 to 3;
         p = p + x[j] * b[j];
         end;

      * Exponentiate x * b and sum them up;
      p   = exp(p);
      sum = sum + p;

      * Output sum exp(x * b) to the macro variable '&sum';
      if eof then call symputx('sum', sum);
      run;

   proc format;
      value df 1 = 'Dark' 0 = 'Milk';
      value sf 1 = 'Soft' 0 = 'Chewy';
      value nf 1 = 'Nuts' 0 = 'No Nuts';
      run;

   * Divide each exp(x * b) by sum exp(x * b);
   data p;
      set p;
      p = p / (&sum);
      format dark df. soft sf. nuts nf.;
      run;

   proc sort;
      by descending p;
      run;

   proc print;
     run;
```

The results are as follows:

```
                        Choice of Chocolate Candies

                 Obs    Dark    Soft     Nuts           p

                  1     Dark    Chewy    Nuts       0.50400
                  2     Dark    Chewy    No Nuts    0.21600
                  3     Milk    Chewy    Nuts       0.12600
                  4     Dark    Soft     Nuts       0.05600
                  5     Milk    Chewy    No Nuts    0.05400
                  6     Dark    Soft     No Nuts    0.02400
                  7     Milk    Soft     Nuts       0.01400
                  8     Milk    Soft     No Nuts    0.00600
```

The three most preferred alternatives are Dark/Chewy/Nuts, Dark/Chewy/No Nuts, and Milk/Chewy/Nuts.

Fabric Softener Example

In this example, subjects are asked to choose among fabric softeners. This example shows all of the steps in a discrete choice study, including experimental design creation and evaluation, creating the questionnaire, inputting the raw data, creating the data set for analysis, coding, fitting the discrete choice model, interpretation, and probability of choice. In addition, custom questionnaires are discussed. We assume that you are familiar with the experimental design issues that are discussed starting on page 53.

Set Up

The study involves four fictitious fabric softeners *Sploosh*, *Plumbbob*, *Platter*, and *Moosey*.[*] There are 50 subjects, each of which see the same choice sets.

Each choice set consists of each of these four brands and a constant alternative *Another*. Each of the brands is available at three prices, $1.49, $1.99, and $2.49. *Another* is only offered at $1.99. The total number of alternatives for this study is four, since there are four brands, and each consists of one attribute, namely price. Since the constant alternative does not vary, it does not enter into the experimental design at this stage. We use an approach in this example that is discussed in detail in the experimental design chapter starting on page 53. Namely, we start by making a "linear arrangement of a choice design," with one factor for every attribute of every alternative, and use that to make our final choice design.

We can use the `%MktRuns` autocall macro to help us choose the number of choice sets. All of the autocall macros used in this book are documented starting on page 803. To use this macro, you specify the number of levels for each of the factors. With four price factors (one for each of the four brands) each with three prices, you specify four 3's (or equivalently, `3 ** 4`). The `%MktRuns` macro is invoked as follows:

```
title 'Choice of Fabric Softener';

%mktruns(3 3 3 3)
```

The output from this macro is as follows:

```
                         Choice of Fabric Softener

                              Design Summary

                          Number of
                          Levels       Frequency

                             3            4
```

[*]Of course real studies use real brands. We have not collected real data, so we cannot use real brand names with artificial data. We picked these silly names so no one would confuse our artificial data with real data.

```
                      Choice of Fabric Softener

          Saturated      = 9
          Full Factorial = 81

          Some Reasonable                        Cannot Be
             Design Sizes        Violations    Divided By

                      9 *S              0
                     18 *               0
                     12                 6    9
                     15                 6    9
                     10                10    3 9
                     11                10    3 9
                     13                10    3 9
                     14                10    3 9
                     16                10    3 9
                     17                10    3 9

    * - 100% Efficient design can be made with the MktEx macro.
    S - Saturated Design - The smallest design that can be made.

                      Choice of Fabric Softener

             n    Design                             Reference

             9             3 **  4              Fractional-Factorial
            18    2 **  1  3 **  7              Orthogonal Array
            18             3 **  6   6 **  1    Orthogonal Array
```

The output first tells us that we specified a design with four factors, each with three levels. The next table reports the size of the saturated design, which is the number of parameters in the linear model based on this design. After that, design sizes are suggested.

The output from this macro tells us that the saturated design has nine runs and the full-factorial design has 81 runs. It also tells us that 9 and 18 are optimal design sizes with zero violations. The macro tells us that in nine runs, an orthogonal design with 4 three-level factors is available, and in 18 runs, two orthogonal and balanced designs are available: one with a two-level factor and 7 three-level factors, and one with 6 three-level factors and a six-level factor. There are zero violations with these designs because these sizes can be divided by 3 and $3 \times 3 = 9$. Twelve and 15 are also reported as potential design sizes, but each has 6 violations. Six times (the $4(4-1)/2 = 6$ pairs of the four 3's) the design sizes of 12 and 15 cannot be divided by $3 \times 3 = 9$. Ideally, we would like to have a manageable number of choice sets for people to evaluate and a design that is both orthogonal and balanced. When violations are reported, orthogonal and balanced designs are not possible. While orthogonality and balance are not required, they are nice properties to have. With 4 three-level factors, the number of choice sets in all orthogonal and balanced designs must be divisible by $3 \times 3 = 9$.

Nine choice sets is a bit small. Furthermore, there are no error *df*. We set the number of choice sets to 18 since it is small enough for each person to see all choice sets, large enough to have reasonable error *df*, and an orthogonal and balanced design is available. It is important to understand, however, that the concept of number of parameters and error *df* discussed here applies to the linear arrangement of the choice design and not to the choice model.* We could use the nine-run design for a discrete choice model and have error *df* in the choice model. If we were to instead use this design for a full-profile conjoint (not recommended), there would be no error *df*.

To make the code easier to modify for future use, the number of choice sets and alternatives are stored in macro variables and the prices are put into a format. Our design, in raw form, has values for price of 1, 2, and 3. We use a format to assign the actual prices: $1.49, $1.99, and $2.49. The format also creates a price of $1.99 for missing, which is used for the constant alternative. The following statements create the macro variables and format:

```
%let n = 18;                    /* n choice sets                    */
%let m = 5;                     /* m alternative including constant */
%let mm1 = %eval(&m - 1);       /* m - 1                            */

proc format;                    /* create a format for the price    */
   value price 1 = '$1.49' 2 = '$1.99' 3 = '$2.49' . = '$1.99';
   run;
```

Designing the Choice Experiment

The next step creates an efficient experimental design. We use the autocall macro `%MktEx` to create many of our designs. (All of the autocall macros used in this book are documented starting on page 803.) When you invoke the `%MktEx` macro for a simple problem, you only need to specify the numbers of levels and the number of runs. The macro does the rest. The `%MktEx` macro usage for this example is as follows:

```
%mktex(3 ** 4, n=&n)
```

The syntax '*n* ** *m*' means *m* factors each at *n* levels. This example has four factors, `x1` through `x4`, all with three levels. The `n=` option specifies the number of runs. The specification `n=&n` is equivalent to `n=18`, and it requests a design in 18 runs. These are all the options that are needed for a simple problem such as this one. However, throughout this book, random number seeds are explicitly specified with the `seed=` option so that the results are reproducible.* The macro call with the random number seed specified is as follows:

```
%mktex(3 ** 4, n=&n, seed=17)
```

*See page 67 for an explanation of the linear arrangement of a choice design versus the arrangement of a choice design that is more suitable for analysis.

*By specifying a random number seed, results should be reproducible within a SAS release for a particular operating system and for a particular version of the macro. However, due to machine and macro differences, some results might not be exactly reproducible everywhere. For most orthogonal and balanced designs, the results should be reproducible. When computerized searches are done, it is likely that you will not get the same design as the one in the book, although you would expect the efficiency differences to be slight.

The results are as follows:

```
                          Choice of Fabric Softener

                           Algorithm Search History

                                  Current          Best
              Design    Row,Col  D-Efficiency  D-Efficiency  Notes
              ----------------------------------------------------------
                   1      Start      100.0000      100.0000  Tab
                   1        End      100.0000
                          Choice of Fabric Softener

                             The OPTEX Procedure

                           Class Level Information

                            Class  Levels  Values

                            x1       3     1 2 3
                            x2       3     1 2 3
                            x3       3     1 2 3
                            x4       3     1 2 3


                          Choice of Fabric Softener

                             The OPTEX Procedure

                                                                     Average
                                                                   Prediction
       Design                                                       Standard
       Number     D-Efficiency     A-Efficiency     G-Efficiency       Error
       ------------------------------------------------------------------------
            1         100.0000         100.0000         100.0000        0.7071
```

The following step displays the design:

```
proc print data=design; run;
```

The design is as follows:

```
                        Choice of Fabric Softener

                       Obs    x1    x2    x3    x4

                         1     1     1     1     1
                         2     1     1     2     2
                         3     1     2     1     3
                         4     1     2     3     1
                         5     1     3     2     3
                         6     1     3     3     2
                         7     2     1     1     3
                         8     2     1     3     1
                         9     2     2     2     2
                        10     2     2     3     3
                        11     2     3     1     2
                        12     2     3     2     1
                        13     3     1     2     3
                        14     3     1     3     2
                        15     3     2     1     2
                        16     3     2     2     1
                        17     3     3     1     1
                        18     3     3     3     3
```

The macro found a perfect, orthogonal and balanced, 100% D-efficient design consisting of 4 three-level factors, `x1-x4`. The levels are the integers 1 to 3. For this problem, the macro generated the design directly. For other problems, the macro might have to use a computerized search. See page 347 for more information about how the `%MktEx` macro works.

Examining the Design

You should run basic checks on all designs, even orthogonal designs such as this one. You can use the `%MktEval` macro to display information about the design. The macro first displays a matrix of canonical correlations between the factors. We hope to see an identity matrix (a matrix of ones on the diagonal and zeros everywhere else) which means the design is orthogonal. Next, the macro displays all one-way frequencies for all attributes, all two-way frequencies, and all n-way frequencies (in this case four-way frequencies). We hope to see equal or at least nearly equal one-way and two-way frequencies, and we want to see that each combination occurs only once. The following step creates the design:

```
   %mkteval(data=design)
```

The results are as follows:

```
                    Choice of Fabric Softener
              Canonical Correlations Between the Factors
         There are 0 Canonical Correlations Greater Than 0.316

                         x1      x2      x3      x4

                   x1    1       0       0       0
                   x2    0       1       0       0
                   x3    0       0       1       0
                   x4    0       0       0       1

                    Choice of Fabric Softener
                      Summary of Frequencies
         There are 0 Canonical Correlations Greater Than 0.316

                         Frequencies

                x1       6 6 6
                x2       6 6 6
                x3       6 6 6
                x4       6 6 6
                x1 x2    2 2 2 2 2 2 2 2 2
                x1 x3    2 2 2 2 2 2 2 2 2
                x1 x4    2 2 2 2 2 2 2 2 2
                x2 x3    2 2 2 2 2 2 2 2 2
                x2 x4    2 2 2 2 2 2 2 2 2
                x3 x4    2 2 2 2 2 2 2 2 2
                N-Way    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

A *canonical correlation* is the maximum correlation between linear combinations of the coded factors (see page 101). All zeros off of the diagonal show that this design is orthogonal for main effects. If any off-diagonal canonical correlations are greater than 0.316 ($r^2 > 0.1$), the macro lists them in a separate table. The last title line tells you that none of them is this large. The criterion $r > 0.316$ or ($r^2 > 0.1$) is purely arbitrary, and you can specify any other value you choose by using the `list=` option.

For nonorthogonal designs and designs with interactions, the canonical-correlation matrix is not a substitute for looking at the variance matrix (with `examine=v`, discussed on pages 351, 425, and 1058). It just provides a quick and more-compact picture of the correlations between the factors. The variance matrix is sensitive to the actual model specification and the actual coding. The canonical-correlation matrix just tells you if there is some correlation between the main effects. In this case, there are no correlations.

The equal one-way frequencies show you that this design is balanced. The equal two-way frequencies show you that this design is orthogonal. The n-way frequencies, all equal to one, show there are no duplicate profiles. This is a perfect design for a main-effects model.

You should always check the n-way frequencies to ensure that you do not have duplicates. For this situation for example, a 100% D-efficient design exists where each run appears twice. It consists of two copies of the fractional-factorial design 3^4 in 9 runs. When you get duplicates, specify `options=nodups` in the `%MktEx` macro, or sometimes you can just change the random number seed. Most designs do not have duplicates, so it is better to specify `options=nodups` only after you have found a design with duplicates. The no-duplicates constraint greatly slows down the algorithm.

The `%MktEval` macro produces a very compact summary of the design, hence some information, for example, the levels to which the frequencies correspond, is not shown. You can use the `print=freqs` option to get a less compact and more detailed display as follows:

```
   %mkteval(data=design, print=freqs)
```

Portions of the results are as follows:

```
                    Choice of Fabric Softener
                           Frequencies

            Effects    Frequency    x1    x2    x3    x4

             x1            6         1     .     .     .
                           6         2     .     .     .
                           6         3     .     .     .

             x2            6         .     1     .     .
                           6         .     2     .     .
                           6         .     3     .     .
            .
            .
            .

             x1 x2         2         1     1     .     .
                           2         1     2     .     .
                           2         1     3     .     .
                           2         2     1     .     .
                           2         2     2     .     .
                           2         2     3     .     .
                           2         3     1     .     .
                           2         3     2     .     .
                           2         3     3     .     .
            .
            .
            .
```

```
x3 x4          2         .     .     1     1
               2         .     .     1     2
               2         .     .     1     3
               2         .     .     2     1
               2         .     .     2     2
               2         .     .     2     3
               2         .     .     3     1
               2         .     .     3     2
               2         .     .     3     3

N-Way          1         1     1     1     1
               1         1     1     2     2
               1         1     2     1     3
               1         1     2     3     1
               1         1     3     2     3
               1         1     3     3     2
               1         2     1     1     3
               1         2     1     3     1
               1         2     2     2     2
               1         2     2     3     3
               1         2     3     1     2
               1         2     3     2     1
               1         3     1     2     3
               1         3     1     3     2
               1         3     2     1     2
               1         3     2     2     1
               1         3     3     1     1
               1         3     3     3     3
```

The Randomized Design and Postprocessing

The design we just looked at and examined is in the default output data set, `Design`. The `Design` data set is the last data set created by the `%MktEx` macro, so it is the data set that you see if you run PROC PRINT or the `%MktEval` macro without specifying the `data=` option. The `Design` data set is sorted, and often the first row consists entirely of ones. For these reasons, you should actually use the *randomized* design. In the randomized design, the choice sets are presented in a random order and the levels have been randomly reassigned. Neither of these operations affects the design *D*-efficiency, balance, or orthogonality, so the `%MktEval` results from the `Design` data set are valid, even if you ultimately use the `Randomized` data set. The macro automatically randomizes the design and stores the results in a data set called `Randomized`. The next steps assign formats and labels and store the results in a SAS data set `sasuser.SoftenerLinDes`. It is important to store the design in a permanent SAS data set or in some other permanent form so that it is available for analysis after the data are collected.

Every SAS data set has a two-level name of the form `libref.filename`. You can always reference a file with its two-level name. However, you can also use a one-level name, and then that data set is stored in temporary SAS data library with a libref of `Work`. Temporary data sets are deleted at the end of your SAS session, so any data that must be saved needs to be stored in a permanent SAS data set. The `libref` called `sasuser` is automatically available for permanent storage in most SAS installations. Furthermore, you can make your own `libref` using a `libname` statement. You might wish to create a separate library for each project. The latter approach of using a `libname` statement is usually preferable, but for our purposes, mainly to avoid discussing issues of host-specific paths and file names, we use `sasuser`. See your BASE SAS documentation and SAS Companion for your operating system for more information about data libraries, `libref`, and `libname`. The following step creates a permanent SAS data set:

```
data sasuser.SoftenerLinDes;
   set randomized;
   format x1-x&mm1 price.;
   label x1 = 'Sploosh' x2 = 'Plumbbob' x3 = 'Platter' x4 = 'Moosey';
   run;
```

The following step displays the final design:

```
proc print data=sasuser.SoftenerLinDes label; /* print final design */
   title2 'Efficient Design';
   run;
```

The results are as follows:

```
                     Choice of Fabric Softener
                          Efficient Design

           Obs    Sploosh    Plumbbob    Platter    Moosey

             1     $1.99      $1.99       $1.99      $2.49
             2     $2.49      $1.49       $1.49      $1.99
             3     $1.49      $2.49       $2.49      $1.49
             4     $2.49      $1.99       $2.49      $1.99
             5     $1.49      $1.49       $1.49      $2.49
             6     $1.49      $2.49       $1.99      $1.99
             7     $2.49      $1.99       $1.99      $1.49
             8     $2.49      $2.49       $1.49      $1.49
             9     $1.99      $1.49       $2.49      $1.49
            10     $1.49      $1.49       $1.99      $1.49
            11     $1.99      $2.49       $1.49      $2.49
            12     $1.49      $1.99       $1.49      $1.99
            13     $1.99      $1.99       $1.49      $1.49
            14     $1.49      $1.99       $2.49      $2.49
            15     $2.49      $1.49       $2.49      $2.49
            16     $1.99      $2.49       $2.49      $1.99
            17     $1.99      $1.49       $1.99      $1.99
            18     $2.49      $2.49       $1.99      $2.49
```

From the Linear Arrangement to a Choice Design

The randomized design is now in a useful form for making the questionnaire, which is discussed in the next section. However, it is not in the final choice-design form that is needed for analysis and for the last evaluation that we should perform before collecting data. In this section, we convert our linear arrangement to a choice design and evaluate its goodness for a choice model.

Our linear arrangement, which we stored in a permanent SAS data set, `sasuser.SoftenerLinDes`, is arranged with one row per choice set. For analysis, we need a *choice design* with one row for each alternative of each choice set. We call the randomized design a *linear arrangement* (see page 67) because we used the `%MktEx` macro to create it optimizing *D*-efficiency for a linear model. We use the macro `%MktRoll` to "roll out" the linear arrangement into the choice design, which is in the proper form for analysis. First, we must create a data set that describes how the design is processed. We call this data set the *design key*.

In this example, we want a choice design with two factors, `Brand` and `Price`. `Brand` has levels *Sploosh*, *Plumbbob*, *Platter*, *Moosey*, and *Another*. `Price` has levels $1.49, $1.99, and $2.49. `Brand` and `Price` are created by different processes. The `Price` attribute is constructed from the factors of the linear arrangement matrix. In contrast, there is no `Brand` factor in the linear arrangement. Each brand is a bin into which its factors are collected. The variable `Brand` is named in the `alt=` option of the `%MktRoll` macro as the alternative variable, so its values are read directly out of the `Key` data set. `Price` is not named in the `alt=` macro option, so its values in the `Key` data set are variable names from the linear arrangement data set. The values of `Price` in the final choice design are read from the named variables in the linear arrangement data set. The `Price` attribute in the choice design is created from the four linear arrangement factors (`x1` for *Sploosh*, `x2` for *Plumbbob*, `x3` for *Platter*, `x4` for *Moosey*, and no attribute for *Another*, the constant alternative). The `Key` data set is created in the next step. The `Brand` factor levels and the `Price` linear arrangement factors are stored in the `Key` data set as follows:

```
title2 'Key Data Set';

data key;
   input Brand $ Price $;
   datalines;
Sploosh   x1
Plumbbob  x2
Platter   x3
Moosey    x4
Another   .
;

proc print; run;
```

The results are as follows:

```
                    Choice of Fabric Softener
                         Key Data Set

                     Obs     Brand      Price

                      1     Sploosh      x1
                      2     Plumbbob     x2
                      3     Platter      x3
                      4     Moosey       x4
                      5     Another
```

Note that the value of `Price` for alternative *Another* in the `Key` data set is blank (character missing). The period in the in-stream data set is simply a place holder, used with list input to read both character and numeric missing data. A period is not stored with the data. Next, we use the `%MktRoll` macro to process the design as follows:

```
%mktroll(design=sasuser.SoftenerLinDes, key=key, alt=brand,
         out=sasuser.SoftenerChDes)
```

The `%MktRoll` step processes the `design=sasuser.SoftenerLinDes` linear arrangement data set using the rules specified in the `key=key` data set, naming the `alt=brand` variable as the alternative name variable, and creating an output SAS data set called `out=sasuser.SoftenerChDes`, which contains the choice design. The input `design=sasuser.SoftenerLinDes` data set has 18 observations, one per choice set, and the output `out=sasuser.SoftenerChDes` data set has $5 \times 18 = 90$ observations, one for each alternative of each choice set. The following step displays the first three observations of the linear arrangement data set:

```
title2 'Linear Arrangement (First 3 Sets)';

proc print data=sasuser.SoftenerLinDes(obs=3); run;
```

The results are as follows:

```
                    Choice of Fabric Softener
                Linear Arrangement (First 3 Sets)

              Obs     x1        x2        x3        x4

               1    $1.99     $1.99     $1.99     $2.49
               2    $2.49     $1.49     $1.49     $1.99
               3    $1.49     $2.49     $2.49     $1.49
```

These observations define the first three choice sets.

The following step displays those same observations, arrayed for analysis in the choice design data set:

```
title2 'Choice Design (First 3 Sets)';

proc print data=sasuser.SoftenerChDes(obs=15);
   format price price.;
   id set; by set;
   run;
```

The results are as follows:

```
                    Choice of Fabric Softener
                   Choice Design (First 3 Sets)

                     Set     Brand      Price

                      1     Sploosh     $1.99
                            Plumbbob    $1.99
                            Platter     $1.99
                            Moosey      $2.49
                            Another     $1.99
                      2     Sploosh     $2.49
                            Plumbbob    $1.49
                            Platter     $1.49
                            Moosey      $1.99
                            Another     $1.99
                      3     Sploosh     $1.49
                            Plumbbob    $2.49
                            Platter     $2.49
                            Moosey      $1.49
                            Another     $1.99
```

The choice design data set has a choice set variable `Set`, an alternative name variable `Brand`, and a price variable `Price`. The prices come from the linear arrangement, and the price for *Another* is a constant $1.99. Recall that the prices are assigned by the following format:

```
proc format;                 /* create a format for the price */
   value price 1 = '$1.49' 2 = '$1.99' 3 = '$2.49' . = '$1.99';
   run;
```

Testing the Design Before Data Collection

Collecting data is time consuming and expensive. It is always good practice to make sure that the design works with the most complicated model that we anticipate fitting. The following step evaluates the choice design:

```
title2 'Evaluate the Choice Design';

%choiceff(data=sasuser.SoftenerChDes,/* candidate set of choice sets       */
          init=sasuser.SoftenerChDes(keep=set),  /* select these sets      */
          intiter=0,                /* evaluate without internal iterations */
                                    /* main effects with ref cell coding    */
                                    /* ref level for brand is 'Another'     */
                                    /* ref level for price is $1.99         */
          model=class(brand price / zero='Another' '$1.99') /
                lprefix=0           /* lpr=0 labels created from just levels*/
                cprefix=0%str(;)    /* cpr=0 names created from just levels */
                format price price.,/* trick: format passed in with model   */

          nsets=&n,                 /* number of choice sets                */
          nalts=&m,                 /* number of alternatives               */
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

The `%ChoicEff` macro has two uses. You can use it to search for an efficient choice design, or you can use it to evaluate a choice design including designs that are generated using other methods such as the `%MktEx` and `%MktRoll` macros. It is this latter use that is illustrated here.

The way you check a design like this is to first name it in the `data=` option. This is the candidate set that contains all of the choice sets that we will consider. In addition, the same design is named in the `init=` option. The full specification is `init=sasuser.SoftenerChDes(keep=set)`. Just the variable `Set` is kept. It is used to bring in just the indicated choice sets from the `data=` design, which in this case is all of them. The option `nsets=&n` specifies that there are `&n`=18 choice sets, and `nalts=&m` specifies that there are `&m`=5 alternatives. The option `beta=zero` specifies that we are assuming for design evaluation purposes the null hypothesis that all of the betas or part-worth utilities are zero. You can evaluate the design for other parameter vectors by specifying a list of numbers after `beta=`. This will change the variances and standard errors. We also specify `intiter=0` which specifies zero internal iterations. We use zero internal iterations when we want to evaluate an initial design, but not attempt to improve it. The `model=` option option specifies the model.

The model specification contains everything that appears in the TRANSREG procedure's `model` statement for coding the design. The specification `class(brand price / zero='Another' '$1.99')` names the `brand` and `price` variable as a classification variables and asks for coded variables for every level except `'Another'` for `brand` and '$1.99' for `price`. The levels `'Another'` and `'$1.99'` are the reference levels for the two attributes. In a p-level factor, there are at most $p - 1$ nonzero parameters.

The `lprefix=0` option specifies that when labels are created for the binary variables, zero characters of the original variable name should be used as a prefix. This means that the labels are created only from the level values. For example, `'Sploosh'` and `'Plumbbob'` are created as labels not `'Brand Sploosh'` and `'Brand Plumbbob'`. The `cprefix=0` option specifies that when names are created for the binary variables, zero characters of the original variable name should be used as a prefix. This means that the names are created only from the level values. The `c` in `cprefix` stands for `class`.

The code following the `cprefix=` specification is a bit of a trick. The `%ChoicEff` macro generates a `model` statement for PROC TRANSREG using the specified value like this:

```
model &model;
```

By adding a semicolon, enclosed in `%str( )` and a format statement, we can send a format statement to the PROC TRANSREG coding step. The semicolon must be in the `%str( )` macro function so that it is passed into the macro and is not treated as the end of the macro specification. The following model specification adds these two statements to PROC TRANSREG in the `%ChoicEff` macro:

```
model class(brand price / zero='Another' '$1.99') / lprefix=0 cprefix=0;
format price price.;
```

Alternatively, we could have just created a separate data set and added the format statement that way.

The results are as follows:

```
                    Choice of Fabric Softener
                    Evaluate the Choice Design

                n    Name        Beta    Label

                1    Moosey        0     Moosey
                2    Platter       0     Platter
                3    Plumbbob      0     Plumbbob
                4    Sploosh       0     Sploosh
                5    _1_49         0     $1.49
                6    _2_49         0     $2.49

                    Choice of Fabric Softener
                    Evaluate the Choice Design

          Design   Iteration  D-Efficiency       D-Error
          ----------------------------------------------
               1       0          2.34223 *     0.42694

                    Choice of Fabric Softener
                    Evaluate the Choice Design

                          Final Results

                    Design                  1
                    Choice Sets            18
                    Alternatives            5
                    Parameters              6
                    Maximum Parameters     72
                    D-Efficiency       2.3422
                    D-Error            0.4269
```

```
                     Choice of Fabric Softener
                     Evaluate the Choice Design

          Variable                                          Standard
     n      Name       Label        Variance     DF          Error

     1    Moosey      Moosey         0.72917      1        0.85391
     2    Platter     Platter        0.72917      1        0.85391
     3    Plumbbob    Plumbbob       0.72917      1        0.85391
     4    Sploosh     Sploosh        0.72917      1        0.85391
     5    _1_49       $1.49          0.52083      1        0.72169
     6    _2_49       $2.49          0.52083      1        0.72169
                                                 ==
                                                  6
```

The first table provides the name, specified value, and label for each parameter. The second table is the iteration history. There is just one line in the table since zero internal iterations are requested. The third table summarizes the design. The first design has 18 choice sets, 5 alternatives, a *D*-efficiency of 2.3422 and a *D*-error of 0.4269. *D*-error = 1 / *D*-efficiency. Note that for this evaluation, *D*-efficiency and *D*-error are computed on a scale with an unknown maximum, so unlike the values that come out of the `%MktEx` macro, *D*-efficiency is not on a percentage or zero to 100 scale in this set of results. Later in this example, we change that. For now, the *D*-efficiency is not what really interests us. We are most interested in the final table. It shows the names and labels for the parameters as well as their variances, standard errors, and *df*. We see that the parameters for all four brands have the same standard errors. Similarly, the standard errors for the two prices are the same. They are different for the two attributes since both have a different number of levels. In both sets, however, they are all approximately the same order of magnitude. Sometimes you might see wildly varying parameters. This is usually a sign of a problematic design, perhaps one with too few choice sets for the number of parameters. This design looks good.

It is a really good idea to perform this step before designing the questionnaire and collecting data. Data collection is expensive, so it is good to make sure that the design can be used for the most complicated model that you intend to fit. Much more is said about evaluating the standard errors in the later and more complicated examples.

You can also evaluate the choice design using the standardized orthogonal contrast coding as follows:

```
title2 'Evaluate the Choice Design';
title3 'Standardized Orthogonal Contrast Coding';

%choiceff(data=sasuser.SoftenerChDes,/* candidate set of choice sets       */
          init=sasuser.SoftenerChDes(keep=set),  /* select these sets      */
          intiter=0,                /* evaluate without internal iterations */
                                    /* model with stdz orthogonal coding    */
                                    /* ref level for brand is 'Another'     */
                                    /* ref level for price is $1.99         */
          model=class(brand price / zero='Another' '$1.99' sta) /
                lprefix=0           /* lpr=0 labels created from just levels*/
                cprefix=0%str(;)    /* cpr=0 names created from just levels */
                format price price.,/* trick: format passed in with model   */

          nsets=&n,                 /* number of choice sets                */
          nalts=&m,                 /* number of alternatives               */
          options=relative,         /* display relative D-efficiency        */
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

The `sta`[*] (short for `standorth`) option in the `model=` option is new with SAS 9.2 and requests a standardized orthogonal contrast coding. You must specify this coding if you want to see relative D-efficiency on a 0 to 100 scale. Relative D-efficiency is not displayed by default since it is only meaningful for a limited class of designs. You must request it with `options=relative`.

The last part of the output from the `%ChoicEff` macro's evaluation of the design is as follows:

```
                    Choice of Fabric Softener
                   Evaluate the Choice Design
             Standardized Orthogonal Contrast Coding

                          Final Results

                    Design                   1
                    Choice Sets             18
                    Alternatives             5
                    Parameters               6
                    Maximum Parameters      72
                    D-Efficiency       15.5119
                    Relative D-Eff     86.1774
                    D-Error             0.0645
                    1 / Choice Sets     0.0556
```

[*]This option is first available with SAS 9.2. It will not be recognized, and it will cause an error in earlier SAS releases.

```
                        Choice of Fabric Softener                        9
                        Evaluate the Choice Design
                   Standardized Orthogonal Contrast Coding

          Variable                                          Standard
     n      Name       Label        Variance    DF       Error

     1    Moosey      Moosey       0.055556     1      0.23570
     2    Platter     Platter      0.055556     1      0.23570
     3    Plumbbob    Plumbbob     0.055556     1      0.23570
     4    Sploosh     Sploosh      0.055556     1      0.23570
     5    _1_49       $1.49        0.086806     1      0.29463
     6    _2_49       $2.49        0.086806     1      0.29463
                                               ==
                                                6
```

In the first table, we see *D*-efficiency equals 15.5119 (as opposed to 2.3422 previous). *D*-error is always equal to 1 / *D*-efficiency. If this were a perfect choice design like we can achieve for some generic designs, then *D*-efficiency would equal 18, and the number of choice sets and *D*-error and all of the variances would equal one over the number of choice sets (0.0556). Relative *D*-efficiency equals 100 times *D*-efficiency divided by the number of choice sets. It is 100 for perfect designs. In this case, relative *D*-efficiency = 86.1774, and all of the variances for the price parameters are larger than 0.556. Note the relative *D*-efficiency = 86.1774 provides a pessimistic view of the goodness of this design. This value is computed relative to the value you would get for an unrestricted design (one that does not have a constant alternative). 100% relative *D*-efficiency is not possible with a constant alternative and using the number of choice sets as a base line since all alternatives must vary optimally to achieve 100% efficiency. Furthermore, this design has five alternatives. There are five brands, one for each alternative, which is ideal. Hence, the variances for the brand parameters are at the minimum. However, there are only three prices. Since 3 does not evenly divide the 5 alternatives, there is no way we can achieve 100% efficiency by scaling *D*-efficiency relative to the number of choice sets. Hence, the variances for the price parameters are greater than their minimum. This design looks reasonable because the variances are not too far removed from one over the number of choice sets.

The `%ChoicEff` macro displays the variances. You can display the full matrix of variances and covariances, which the `%ChoicEff` macro stores in a SAS data set, as follows:

```
proc print data=bestcov label;
   title3 'Variance-Covariance Matrix';
   id __label;
   label __label = '00'x;
   var Moosey -- _2_49;
   run;
```

The results are as follows:

```
                              Choice of Fabric Softener
                              Evaluate the Choice Design
                              Variance-Covariance Matrix

               Moosey     Platter     Plumbbob    Sploosh      $1.49       $2.49

   Moosey      0.055556   -0.000000   -0.000000    0.000000    0.000000   -0.000000
   Platter    -0.000000    0.055556    0.000000   -0.000000   -0.000000    0.000000
   Plumbbob   -0.000000    0.000000    0.055556    0.000000   -0.000000    0.000000
   Sploosh     0.000000   -0.000000    0.000000    0.055556    0.000000   -0.000000
   $1.49       0.000000   -0.000000   -0.000000    0.000000    0.086806   -0.000000
   $2.49      -0.000000    0.000000    0.000000   -0.000000   -0.000000    0.086806
```

All of the covariances are zero, which is good. The larger variances are due to the within-choice-set imbalance and perhaps inefficiency in price.

There is one more test that should be run before a design is used. The following `%MktDups` macro step checks the design to see if any choice sets are duplicates of any other choice sets:

```
   title2 'Evaluate the Choice Design, Check for Duplicates';

   %mktdups(branded, data=sasuser.SoftenerChDes, nalts=&m, factors=brand price)
```

The results are as follows:

```
   Design:          Branded
   Factors:         brand price
                    Brand
                    Price
   Duplicate Sets:  0
```

The first line of the table tells us that this is a branded design as opposed to a generic design (bundles of attributes with no brands). The second line tells us the factors as specified in the `factors=` option. These are followed by the actual variable names for the factors. The last line reports the number of duplicates. In this case, there are no duplicate choice sets. When there are duplicate choice sets, changing the random number seed might help. Changing other aspects of the design or the approach for making the design might also help.

Evaluating the Design Relative to the Optimal Design

In the previous section, we saw that our design had a *D*-efficiency = 86.1774 relative to a design with no constant alternative restriction, which provided a pessimistic view of the goodness of this design. This problem is small enough, with only $3^4 = 81$ possible choice sets that we can be reasonably confident that the `%ChoicEff` macro will find *the* optimal design given a candidate set of all possible choice sets. We can then use the *D*-efficiency of the optimal design in the relative *D*-efficiency computations for our design to measure the goodness of our design for this particular model. This is illustrated in this section. It proceeds very similarly to the steps shown previously, only this time, rather than giving the `%ChoicEff` macro a set of choice sets to evaluate, it is given a set of candidate choice sets from which to construct a design. The following steps create a candidate set of all possible choice sets:

```
%mktex(3 ** 4, n=81)

data TestLinDes;
   set design;
   format x1-x&mm1 price.;
   label x1 = 'Sploosh' x2 = 'Plumbbob' x3 = 'Platter' x4 = 'Moosey';
   run;

data key;
   input Brand $ Price $;
   datalines;
Sploosh   x1
Plumbbob  x2
Platter   x3
Moosey    x4
Another   .
;

%mktroll(design=TestLinDes, key=key, alt=brand, out=TestChDes)
```

The following step searches for an optimal design:

```
%choiceff(data=TestChDes,           /* candidate set of choice sets         */
                                    /* model with stdz orthogonal coding    */
                                    /* ref level for brand is 'Another'     */
                                    /* ref level for price is $1.99         */
          model=class(brand price / zero='Another' '$1.99' sta) /
                lprefix=0           /* lpr=0 labels created from just levels*/
                cprefix=0%str(;)    /* cpr=0 names created from just levels */
                format price price.,/* trick: format passed in with model   */

          seed=205,                 /* random number seed                   */
          maxiter=50,               /* maximum number of designs to make    */
          nsets=&n,                 /* number of choice sets                */
          nalts=&m,                 /* number of alternatives               */
          options=relative,         /* display relative D-efficiency        */
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

Some of the results are as follows:

```
                        Choice of Fabric Softener
              Evaluate the Choice Design, Check for Duplicates

                              Final Results

                        Design                 1
                        Choice Sets           18
                        Alternatives           5
                        Parameters             6
                        Maximum Parameters    72
                        D-Efficiency     16.4264
                        Relative D-Eff   91.2581
                        D-Error           0.0609
                        1 / Choice Sets   0.0556

                        Choice of Fabric Softener
              Evaluate the Choice Design, Check for Duplicates

           Variable                                      Standard
     n       Name      Label       Variance    DF       Error

     1     Moosey      Moosey      0.055556     1      0.23570
     2     Platter     Platter     0.055556     1      0.23570
     3     Plumbbob    Plumbbob    0.055556     1      0.23570
     4     Sploosh     Sploosh     0.055556     1      0.23570
     5     _1_49       $1.49       0.073099     1      0.27037
     6     _2_49       $2.49       0.073099     1      0.27037
                                               ==
                                                6
```

The iteration histories for the 50 designs that were created (not shown) all have the same *D*-efficiency = 16.4264. With small problems like this one, this is a good sign that the optimal design was found.

The following step evaluates our design from the previous section using the *D*-efficiency from what is probably the optimal design in the `rscale=` option:

```
%choiceff(data=sasuser.SoftenerChDes,/* candidate set of choice sets       */
          init=sasuser.SoftenerChDes(keep=set),  /* select these sets      */
          intiter=0,                /* evaluate without internal iterations */
          rscale=16.4264,           /* optimal D-efficiency                 */
                                    /* model with stdz orthogonal coding    */
                                    /* ref level for brand is 'Another'     */
                                    /* ref level for price is $1.99         */
          model=class(brand price / zero='Another' '$1.99' sta) /
                lprefix=0           /* lpr=0 labels created from just levels*/
                cprefix=0%str(;)    /* cpr=0 names created from just levels */
                format price price.,/* trick: format passed in with model   */

          nsets=&n,                 /* number of choice sets                */
          nalts=&m,                 /* number of alternatives               */
          options=relative,         /* display relative D-efficiency        */
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

Some of the results are as follows:

```
                              Choice of Fabric Softener
                     Evaluate the Choice Design, Check for Duplicates

                                   Final Results

                              Design                   1
                              Choice Sets             18
                              Alternatives             5
                              Parameters               6
                              Maximum Parameters      72
                              D-Efficiency       15.5119
                              Relative D-Eff     94.4329
                              D-Error             0.0645
                              1 / Choice Sets     0.0556
```

This shows that relative to what is probably the optimal design, our design is 94.4329% *D*-efficient. Of course, usually we cannot know this. We will continue with this example using the original design as planned, although for this particular model and beta specification, the design we just found is a little better. Of course, we do not know the true beta, although we are sure it is *not* our null hypothesis zero vector, so in practice, we do not in fact know which design is most optimal. Our original design is good in that all of the prices are balanced and orthogonal (at least outside the constant alternative) across all alternatives. In contrast, the design we just found is good in a different way since it was optimized for this model and beta specification.

Generating the Questionnaire

A questionnaire based on the design is produced using the following DATA step:

```
title;

data _null_;                    /* print questionnaire */
   array brands[&m] $ _temporary_ ('Sploosh' 'Plumbbob' 'Platter'
                                   'Moosey' 'Another');
   array x[&m] x1-x&m;
   file print linesleft=ll;
   set sasuser.SoftenerLinDes;

   x&m = 2;                     /* constant alternative */
   format x&m price.;

   if _n_ = 1 or ll < 12 then do;
      put _page_;
      put @60 'Subject: _________' //;
      end;

   put _n_ 2. ') Circle your choice of '
       'one of the following fabric softeners:' /;

   do brnds = 1 to &m;
      put '    ' brnds 1. ') ' brands[brnds] 'brand at '
          x[brnds] +(-1) '.' /;
      end;
   run;
```

The following statement creates a constant array:

```
array brands[&m] $ _temporary_ ('Sploosh' 'Plumbbob' 'Platter'
                                'Moosey' 'Another')
```

The array reference `brands[1]` accesses the string `'Sploosh'`, `brands[2]` accesses the string `'Plumbbob'`, and so on. The `_temporary_` specification means that no output data set variables are created for this array. The `linesleft=` specification in the `file` statement creates the variable `ll`, which contains the number of lines left on a page. This ensures that each choice set is not split over two pages.

In the interest of space, only the first two choice sets are shown as follows:

```
                                                   Subject: _________


   1) Circle your choice of one of the following fabric softeners:

      1) Sploosh brand at $1.99.

      2) Plumbbob brand at $1.99.

      3) Platter brand at $1.99.

      4) Moosey brand at $2.49.

      5) Another brand at $1.99.

   2) Circle your choice of one of the following fabric softeners:

      1) Sploosh brand at $2.49.

      2) Plumbbob brand at $1.49.

      3) Platter brand at $1.49.

      4) Moosey brand at $1.99.

      5) Another brand at $1.99.
```

The questionnaire is printed, copied, and the data are collected.

In practice, data collection is typically much more elaborate than this. It might involve art work or photographs, and the choice sets might be presented and the data might be collected through personal interview or over the Web. However the choice sets are presented and the data are collected, the essential elements remain the same. Subjects are shown a set of alternatives and are asked to make a choice, then they go on to the next set.

Entering the Data

The data consist of a subject number followed by 18 integers in the range 1 to 5. These are the alternatives that are chosen for each choice set. For example, the first subject chose alternative 3 (*Platter* brand at $1.99) in the first choice set, alternative 3 (*Platter* brand at $1.49) in the second choice set, and so on. In the interest of space, data from three subjects appear on one line. The data are read in the following step:

```
title 'Choice of Fabric Softener';

data results;                    /* read choice data set */
   input Subj (choose1-choose&n) (1.) @@;
   datalines;
  1 334533434233312433  2 334213442433333325  3 333333333333313333
  4 334431444434412453  5 335431434233512423  6 334433434433312433
  7 334433434433322433  8 334433434433412423  9 334433332353312433
 10 325233435233332433 11 334233434433313333 12 334331334433312353
 13 534333334333312323 14 134421444433412423 15 334333435433312335
 16 334433435333315333 17 534333432453312423 18 334435544433412543
 19 334333335433313433 20 331431434233315533 21 334353534433512323
 22 334333452233312523 23 334333332333312433 24 525221444233322423
 25 354333434433312333 26 334435545233312323 27 334353534233352323
 28 334333332333332333 29 334433534335352423 30 334453434533313433
 31 354333334333312433 32 354331332233332423 33 334424432353312325
 34 334433434433312433 35 334551444453412325 36 334234534433312433
 37 334431434433512423 38 354333334433352523 39 334351334333312533
 40 324433334433412323 41 334433444433412443 42 334433434433312423
 43 334434454433332423 44 334433434233312423 45 334451544433412424
 46 434431435433512423 47 524434534433412433 48 335453334433322453
 49 334533434133312433 50 334433332333312423
;
```

Processing the Data

The next step merges the choice data with the choice design using the `%MktMerge` macro:

```
proc format;
   value price 1 = '$1.49' 2 = '$1.99' 3 = '$2.49' . = '$1.99';
   run;

%mktmerge(design=sasuser.SoftenerChDes, data=results, out=res2,
          nsets=&n, nalts=&m, setvars=choose1-choose&n)
```

This step reads the `design=sasuser.SoftenerChDes` choice design and the `data=results` data set and creates the `out=res2` output data set. The data are from an experiment with `nsets=&n` choice sets, `nalts=&m` alternatives, with variables `setvars=choose1-choose&n` containing the numbers of the chosen alternatives. The following step displays the first 15 observations:

```
title2 'Choice Design and Data (First 3 Sets)';

proc print data=res2(obs=15);
   id subj set; by subj set;
   run;
```

The results are as follows:

```
                    Choice of Fabric Softener
               Choice Design and Data (First 3 Sets)

               Subj    Set     Brand       Price    c

                 1      1     Sploosh        2      2
                              Plumbbob       2      2
                              Platter        2      1
                              Moosey         3      2
                              Another        .      2

                 1      2     Sploosh        3      2
                              Plumbbob       1      2
                              Platter        1      1
                              Moosey         2      2
                              Another        .      2

                 1      3     Sploosh        1      2
                              Plumbbob       3      2
                              Platter        3      2
                              Moosey         1      1
                              Another        .      2
```

The data set contains the subject ID variable `Subj` from the `data=results` data set, the `Set`, `Brand`, and `Price` variables from the `design=sasuser.SoftenerChDes` data set, and the variable `c`, which indicates which alternative is chosen. The variable `c` contains: 1 for first choice and 2 for second or subsequent choice. This subject chose the third alternative, *Platter* in the first choice set, *Platter* in the second, and *Moosey* in the third. This data set has 4500 observations: 50 subjects times 18 choice sets times 5 alternatives.

Since we did not specify a format, we see in the design the raw design values for `Price`: 1, 2, 3 and missing for the constant alternative. If we were going to treat `Price` as a categorical variable for analysis, this would be fine. We would simply assign our price format to `Price` and designate it as a `class` variable. However, in this analysis we are going to treat price as quantitative and use the actual prices in the analysis. Hence, we must convert our design values from 1, 2, 3, and . to 1.49, 1.99, 2.49, and 1.99. We cannot do this by simply assigning a format. Formats create character strings that are displayed in place of the original value. We need to convert a numeric variable from one set of numeric values to another. We could use `if` and assignment statements. We could also use the `%MktLab` macro, which is used in later examples. However, instead we can use the `put` function to write the formatted value into a character string, then we read it back using a dollar format and the `input` function. For example, the expression `put(price, price.)` converts a number, say 2, into a string (in this case '$1.99'), then the `input` function reads the string and converts it to a numeric 1.99. The following step also assigns a label to the variable `Price`:

```
data res3;   /* Create a numeric actual price */
   set res2;
   price = input(put(price, price.), dollar5.);
   label price = 'Price';
   run;
```

Binary Coding

One more thing must be done to these data before they can be analyzed. The factors must be coded. In this example, we use a *binary* or zero-one coding for the brand effect. This can be done with PROC TRANSREG as follows:

```
proc transreg design=5000 data=res3 nozeroconstant norestoremissing;
   model class(brand / zero=none order=data)
         identity(price) / lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   id subj set c;
   run;
```

The `design` option specifies that no model is fit; the procedure is just being used to code a design. When `design` is specified, dependent variables are not required. Optionally, `design` can be followed by "=*n*" where *n* is the number of observations to process at one time. By default, PROC TRANSREG codes all observations in one big group. For very large data sets, this can consume large amounts of memory and time. Processing blocks of smaller numbers of observations is more computationally efficient. The option `design=5000` processes observations in blocks of 5000. For smaller computers, try something like `design=1000`.

The `nozeroconstant` and `norestoremissing` options are not necessary for this example, but they are included here, because sometimes they are very helpful in coding choice models. The `nozeroconstant` option specifies that if the coding creates a constant variable, it should not be zeroed. The `nozeroconstant` option should always be specified when you specify `design=`*n* because the last group of observations might be small and might contain constant variables. The `nozeroconstant` option is also important if you do something like coding `by subj set` because sometimes an attribute is constant within a choice set. The `norestoremissing` option specifies that missing values should not be restored when the `out=` data set is created. By default, the coded `class` variable contains a row of missing values for observations in which the `class` variable is missing. When you specify the `norestoremissing` option, these observations contain a row of zeros instead. This option is useful when there is a constant alternative indicated by missing values. Both of these options, like almost all options in PROC TRANSREG, can be abbreviated to three characters (`noz` and `nor`).

The `model` statement names the variables to code and provides information about how they should be coded. The specification `class(brand / ...)` specifies that the variable `Brand` is a classification variable and requests a binary coding. The `zero=none` option creates binary variables for all categories. In contrast, by default, a binary variable is not created for the last category—the parameter for the last category is a structural zero. The `zero=none` option is used when there are no structural zeros or when you want to see the structural zeros in the multinomial logit parameter estimates table. See page 78 for more information about the `zero=` option. The `order=data` option sorts the levels into the order that they are first encountered in the data set. Alternatively, the levels could be sorted based on the formatted or unformatted values. The specification `identity(price)` specifies that `Price` is a quantitative attribute that should be analyzed as is (not expanded into indicator variables).

The `lprefix=0` option specifies that when labels are created for the binary variables, zero characters of the original variable name should be used as a prefix.

An `output` statement names the output data set and drops variables that are not needed. These variables do not have to be dropped. However, since they are variable names that are often found in special data set types, PROC PHREG displays warnings when it finds them. Dropping the variables prevents the warnings. Finally, the `id` statement names the additional variables that we want copied from the input to the output data set. The next steps display the first three coded choice sets:

```
proc print data=coded(obs=15) label;
   title2 'First 15 Observations of Analysis Data Set';
   id subj set c;
   by subj set;
   run;
```

The results are as follows:

```
                          Choice of Fabric Softener
                  First 15 Observations of Analysis Data Set

Subj  Set  c  Sploosh  Plumbbob  Platter  Moosey  Another  Price   Brand

  1    1   2     1         0        0        0       0      1.99  Sploosh
           2     0         1        0        0       0      1.99  Plumbbob
           1     0         0        1        0       0      1.99  Platter
           2     0         0        0        1       0      2.49  Moosey
           2     0         0        0        0       1      1.99  Another

  1    2   2     1         0        0        0       0      2.49  Sploosh
           2     0         1        0        0       0      1.49  Plumbbob
           1     0         0        1        0       0      1.49  Platter
           2     0         0        0        1       0      1.99  Moosey
           2     0         0        0        0       1      1.99  Another

  1    3   2     1         0        0        0       0      1.49  Sploosh
           2     0         1        0        0       0      2.49  Plumbbob
           2     0         0        1        0       0      2.49  Platter
           1     0         0        0        1       0      1.49  Moosey
           2     0         0        0        0       1      1.99  Another
```

Fitting the Multinomial Logit Model

The next step fits the discrete choice, multinomial logit model:

```
proc phreg data=coded outest=betas brief;
   title2 'Discrete Choice Model';
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;
```

The `brief` option requests a brief summary for the strata. As with the candy example, `c*c(2)` designates the chosen and unchosen alternatives in the `model` statement. We specify the `&_trgind` macro variable for the `model` statement independent variable list. PROC TRANSREG automatically creates this macro variable. It contains the list of coded independent variables generated by the procedure. This is so you do not have to figure out what names TRANSREG created and specify them. In this case, PROC TRANSREG sets `&_trgind` to contain the following list:

```
BrandSploosh BrandPlumbbob BrandPlatter BrandMoosey BrandAnother Price
```

The `ties=breslow` option specifies a PROC PHREG model that has the same likelihood as the multinomial logit model for discrete choice. The `strata` statement specifies that the combinations of `Set` and `Subj` indicate the choice sets. This data set has 4500 observations consisting of $18 \times 50 = 900$ strata and five observations per stratum.

Each subject rated 18 choice sets, but the multinomial logit model assumes each stratum is independent. That is, the multinomial logit model assumes each person makes only one choice. The option of collecting only one datum from each subject is too expensive to consider for many problems, so multiple choices are collected from each subject, and the repeated measures aspect of the problem is ignored. This practice is typical, and it usually works well, because the parameter estimates are still unbiased.

Multinomial Logit Model Results

Recall that we specified `%phchoice(on)` on page 287 to customize the output from PROC PHREG. The results are as follows:

```
                         Choice of Fabric Softener
                           Discrete Choice Model

                            The PHREG Procedure

                             Model Information

                   Data Set                  WORK.CODED
                   Dependent Variable        c
                   Censoring Variable        c
                   Censoring Value(s)        2
                   Ties Handling             BRESLOW
```

```
                    Number of Observations Read       4500
                    Number of Observations Used       4500

        Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

              Number of       Number of          Chosen          Not
    Pattern     Choices    Alternatives    Alternatives       Chosen

          1         900               5               1            4

                             Convergence Status

               Convergence criterion (GCONV=1E-8) satisfied.

                           Model Fit Statistics

                                  Without          With
                    Criterion     Covariates     Covariates

                    -2 LOG L        2896.988       1439.457
                    AIC             2896.988       1449.457
                    SBC             2896.988       1473.469

                    Testing Global Null Hypothesis: BETA=0

        Test                Chi-Square       DF     Pr > ChiSq

        Likelihood Ratio     1457.5310        5         <.0001
        Score                1299.7889        5         <.0001
        Wald                  635.9093        5         <.0001

                          Choice of Fabric Softener
                            Discrete Choice Model

                             The PHREG Procedure

                     Multinomial Logit Parameter Estimates

                       Parameter      Standard
              DF        Estimate         Error    Chi-Square    Pr > ChiSq

  Sploosh      1        -1.30565       0.21097       38.3017        <.0001
  Plumbbob     1        -0.49090       0.18035        7.4091        0.0065
  Platter      1         2.08485       0.14592      204.1402        <.0001
  Moosey       1         0.62183       0.15503       16.0884        <.0001
  Another      0               0             .             .             .
  Price        1        -4.60150       0.21608      453.5054        <.0001
```

The procedure output begins with information about the data set, variables, options, and number of observations read. This is followed by information about the 900 strata. Since the `brief` option is specified, this table contains one row for each stratum pattern. In contrast, the default table has 900 rows, one for each choice set and subject combination. Each subject and choice set combination consists of a total of five observations, one that is chosen and four that are not chosen. This pattern is observed 900 times. This table provides a check on data entry. Unless we have an availability or allocation study (page 524) or a nonconstant number of alternatives in different choice sets, we expect to see one pattern of results where one of the m alternatives is chosen for each choice set. If you do not observe this for a study like this, there is probably a mistake in the data entry or processing.

The most to least preferred brands are: *Platter*, *Moosey*, *Another*, *Plumbbob*, and *Sploosh*. Increases in price have a negative utility. For example, the predicted utility of *Platter* brand at \$1.99 is $\mathbf{x}_i\boldsymbol{\beta}$ which is $(0 \quad 0 \quad 1 \quad 0 \quad 0 \quad \$1.99) \quad (-1.31 \quad -0.49 \quad 2.08 \quad 0.62 \quad 0 \quad -4.60)' = 2.08 + 1.99 \times -4.60 = -7.07$ Since `Price` is analyzed as a quantitative attribute, we can see for example that the utility of *Platter* at \$1.89, which is not in any choice set, is $2.08 + 1.89 \times -4.60 = -6.61$, which is a $\$0.10 \times 4.60 = 0.46$ increase in utility.

Probability of Choice

These next steps compute the expected probability that each alternative is chosen within each choice set. These steps could easily be modified to compute expected market share for hypothetical marketplaces that do not directly correspond to the choice sets. Note, however, that a term like "expected market share," while widely used, is a misnomer. Without purchase volume data, it is unlikely that these numbers mirror true market share.

First, PROC SCORE is used to compute the predicted utility for each alternative as follows:

```
proc score data=coded(where=(subj=1) drop=c)
           score=betas type=parms out=p;
   var &_trgind;
   run;
```

The data set to be scored is named with the `data=` option, and the coefficients are specified in the option `score=beta`. Note that we only need to read all of the choice sets once, since the parameter estimates are computed in an aggregate analysis. This is why we specified `where=(subj=1)`. We do not need $\mathbf{x}_j\hat{\boldsymbol{\beta}}$ for each of the different subjects. We dropped the variable `c` from the `Coded` data set since this name is used by PROC SCORE for the results ($\mathbf{x}_j\hat{\boldsymbol{\beta}}$). The option `type=parms` specifies that the `score=` data set contains the parameters in `_TYPE_ = 'PARMS'` observations. The output data set with the predicted utilities is named P. Scoring is based on the coded variables from PROC TRANSREG, whose names are contained in the macro variable `&_trgind`. The next step exponentiates $\mathbf{x}_j\hat{\boldsymbol{\beta}}$:

```
data p2;
   set p;
   p = exp(c);
   run;
```

Next, $\exp(\mathbf{x}_j\hat{\boldsymbol{\beta}})$ is summed for each choice set as follows:

```
proc means data=p2 noprint;
   output out=s sum(p) = sp;
   by set;
   run;
```

Finally, each $\mathbf{x}_j\hat{\boldsymbol{\beta}}$ is divided by $\sum_{j=1}^{m}\mathbf{x}_j\hat{\boldsymbol{\beta}}$ in the following step:

```
data p;
   merge p2 s(keep=set sp);
   by set;
   p = p / sp;
   keep brand set price p;
   run;


proc print data=p(obs=15);
   title2 'Choice Probabilities for the First 3 Choice Sets';
   id set; by set;
   run;
```

The results for the first three choice sets are as follows:

```
                      Choice of Fabric Softener
            Choice Probabilities for the First 3 Choice Sets

                  Set    Price     Brand         p

                   1      1.99    Sploosh     0.02680
                          1.99    Plumbbob    0.06052
                          1.99    Platter     0.79535
                          2.49    Moosey      0.01845
                          1.99    Another     0.09888

                   2      2.49    Sploosh     0.00030
                          1.49    Plumbbob    0.06843
                          1.49    Platter     0.89921
                          1.99    Moosey      0.02086
                          1.99    Another     0.01120

                   3      1.49    Sploosh     0.11679
                          2.49    Plumbbob    0.00265
                          2.49    Platter     0.03479
                          1.49    Moosey      0.80260
                          1.99    Another     0.04318
```

Custom Questionnaires

In this part of the example, a custom questionnaire is displayed for each person. Previously, each subject saw the same questionnaire, with the same choice sets, each containing the same alternatives, with everything in the same order. In this example, the order of the choice sets and all alternatives within choice sets are randomized for each subject. Randomizing avoids any systematic effects due to the order of the alternatives and choice sets. The constant alternative is always displayed last. If you have no interest in custom questionnaires, you can skip ahead to page 339.

First, the macro variable `&forms` is created. It contains the number of separate questionnaires (or forms or subjects, in this case 50). We can use the `%MktEx` macro to create a data set with one observation for each alternative of each choice set for each person. The specification `%mktex(&forms &n &mm1, n=&forms * &n * &mm1)` is `%mktex(50 18 4, n=50 * 18 * 4)` and creates a $50 \times 18 \times 4$ full-factorial design. Note that the `n=` specification allows expressions. The macro `%MktLab` is then used to assign the variable names `Form`, `Set`, and `Alt` instead of the default `x1 - x3`. The data set is sorted by `Form`. Within `Form`, the choice sets are sorted into a random order, and within choice set, the alternatives are sorted into a random order. The 72 observations for each choice set contain 18 blocks of 4 observations—one block per choice set in a random order and the 4 alternatives within each choice set, again in a random order. Note that we store these in a permanent SAS data set so they are available after the data are collected. See page 309 for more information about permanent SAS data sets. The following steps create the design:

```
%let forms = 50;
title2 'Create 50 Custom Questionnaires';

*---Make the design---;
%mktex(&forms &n &mm1, n=&forms * &n * &mm1)

*---Assign Factor Names---;
%mktlab(data=design, vars=Form Set Alt)

*---Set up for Random Ordering---;
data sasuser.orders;
   set final;
   by form set;
   retain r1;
   if first.set then r1 = uniform(17);
   r2 = uniform(17);
   run;

*---Random Sort---;
proc sort out=sasuser.orders(drop=r:); by form r1 r2; run;

proc print data=sasuser.orders(obs=16); run;
```

The first 16 observations in this data set are as follows:

```
                         Choice of Fabric Softener
                      Create 50 Custom Questionnaires

                          Obs    Form    Set    Alt

                            1      1       4     3
                            2      1       4     1
                            3      1       4     2
                            4      1       4     4
                            5      1       8     2
                            6      1       8     3
                            7      1       8     1
                            8      1       8     4
                            9      1      16     1
                           10      1      16     2
                           11      1      16     3
                           12      1      16     4
                           13      1       1     3
                           14      1       1     1
                           15      1       1     4
                           16      1       1     2
```

The data set is transposed, so the resulting data set contains $50 \times 18 = 900$ observations, one per subject per choice set. The alternatives are in the variables Col1-Col4. The first 18 observations, which contain the ordering of the choice sets for the first subject, are displayed by the following steps:

```
proc transpose data=sasuser.orders out=sasuser.orders(drop=_name_);
   by form notsorted set;
   run;

proc print data=sasuser.orders(obs=18);
   run;
```

The results are as follows:

```
                         Choice of Fabric Softener
                      Create 50 Custom Questionnaires

              Obs    Form    Set    COL1    COL2    COL3    COL4

                1      1       4      3       1       2       4
                2      1       8      2       3       1       4
                3      1      16      1       2       3       4
                4      1       1      3       1       4       2
                5      1       6      2       4       1       3
                6      1       7      4       1       3       2
```

```
                   7       1      12       3        2        1        4
                   8       1       2       2        4        1        3
                   9       1      17       3        4        1        2
                  10       1      15       4        2        3        1
                  11       1      14       1        2        3        4
                  12       1      10       2        4        3        1
                  13       1       5       1        4        2        3
                  14       1       9       2        4        1        3
                  15       1      13       3        2        1        4
                  16       1       3       3        4        2        1
                  17       1      18       4        2        1        3
                  18       1      11       3        1        4        2
```

The following DATA step displays the 50 custom questionnaires:

```
title;

data _null_;
   array brands[&mm1] $ _temporary_
                ('Sploosh' 'Plumbbob' 'Platter' 'Moosey');
   array x[&mm1] x1-x&mm1;
   array c[&mm1] col1-col&mm1;
   format x1-x&mm1 price.;
   file print linesleft=ll;

   do frms = 1 to &forms;
      do choice = 1 to &n;
         if choice = 1 or ll < 12 then do;
            put _page_;
            put @60 'Subject: ' frms //;
            end;
         put choice 2. ') Circle your choice of '
             'one of the following fabric softeners:' /;
         set sasuser.orders;
         set sasuser.SoftenerLinDes point=set;
         do brnds = 1 to &mm1;
            put '    ' brnds 1. ') ' brands[c[brnds]] 'brand at '
                x[c[brnds]] +(-1) '.' /;
            end;
         put '    5) Another brand at $1.99.' /;
         end;
      end;
   stop;
   run;
```

The loop `do frms = 1 to &forms` creates the 50 questionnaires. The loop `do choice = 1 to &n` creates the alternatives within each choice set. On the first choice set and when there is not enough room for the next choice set, we skip to a new page (`put _page_`) and print the subject (forms) number. The data set `sasuser.Orders` is read and the `Set` variable is used to read the relevant observation from `sasuser.SoftenerLinDes` using the `point=` option in the `set` statement. The order of the alternatives is in the `c` array and variables `col1-col&mm1` from the `sasuser.Orders` data set. In the first observation of `sasuser.Orders`, `Set=4`, `Col1=3`, `Col2=1`, `Col3=2`, and `Col4=4`. The first brand, is `c[brnds] = c[1] = col1 = 3`, so `brands[c[brnds]] = brands[c[1]] = brands[3] = 'Platter'`, and the price, from observation `Set=4` of `sasuser.SoftenerLinDes`, is `x[c[brnds]] = x[3] = $2.49`. The second brand, is `c[brnds] = c[2] = col2 = 1`, so `brands[c[brnds]] = brands[c[2]] = brands[1] = 'Sploosh'`, and the price, from observation `Set=4` of `sasuser.SoftenerLinDes`, is `x[c[brnds]] = x[1] = $2.49`.

In the interest of space, only the first two choice sets are displayed. Note that the subject number is displayed on the form. This information is needed to restore all data to the original order. The first two choice sets are as follows:

```
                                                         Subject: 1


    1) Circle your choice of one of the following fabric softeners:

       1) Platter brand at $2.49.

       2) Sploosh brand at $2.49.

       3) Plumbbob brand at $1.99.

       4) Moosey brand at $1.99.

       5) Another brand at $1.99.

    2) Circle your choice of one of the following fabric softeners:

       1) Plumbbob brand at $2.49.

       2) Platter brand at $1.49.

       3) Sploosh brand at $2.49.

       4) Moosey brand at $1.49.

       5) Another brand at $1.99.
```

Processing the Data for Custom Questionnaires

The data are entered next. (Actually, these are the data that would have been collected if the same people as in the previous situation made the same choices, without error and uninfluenced by order effects.) Before these data are analyzed, the original order must be restored as follows:

```
title 'Choice of Fabric Softener';

data results;                    /* read choice data set */
   input Subj (choose1-choose&n) (1.) @@;
   datalines;
  1 524141141211421241  2 532234223321321311  3 223413221434144231
  4 424413322222544331  5 123324312534444533  6 233114423441143321
  7 123243224422433312  8 312432241121112412  9 315432222144111124
 10 511432445343442414 11 331244123342421432 12 323234114312123245
 13 312313434224435334 14 143433332142334114 15 234423133531441145
 16 425441421454434414 17 234431535341441432 18 235224352241523311
 19 134331342432542243 20 335331253334232433 21 513453254214134224
 22 212241213544214125 23 133444341431414432 24 453424142151142322
 25 324424431252444221 26 244145452131443415 27 553254131423323121
 28 233423242432231424 29 322454324541433543 30 323433433135133542
 31 412422434342513222 32 243144343352123213 33 441113141133454445
 34 131114113312342312 35 325222444355122522 36 342133254432124342
 37 511322324114234222 38 522153113442344541 39 211542232314512412
 40 244432222212213211 41 241411341323123213 42 314334342111232114
 43 422351321313343332 44 124243444234124432 45 141251113314352121
 46 414215225442424413 47 333452434454311222 48 334325341342552344
 49 335124122444243112 50 244412331342433332
;
```

The data set is transposed, and the original order is restored as follows:

```
proc transpose data=results   /* create one obs per choice set */
               out=res2(rename=(col1=choose) drop=_name_);
   by subj;
   run;

data res3(keep=subj set choose);
   array c[&mm1] col1-col&mm1;
   merge sasuser.orders res2;
   if choose < 5 then choose = c[choose];
   run;


proc sort; by subj set; run;
```

The actual choice number, stored in `Choose`, indexes the alternative numbers from `sasuser.Orders` to restore the original alternative orders. For example, for the first subject, the first choice is 5, which is the *Another* constant alternative. Since the first subject saw the fourth choice set first, the fourth data value for the first subject in the processed data set has a value of 5. The choice in the second choice set for the first subject is 2, and the second alternative the subject saw is *Platter*. The data set `sasuser.Orders` shows in the second observation that this choice of 2 corresponds to the third (original) alternative (in the second column variable, `Col2 = 3`) of choice set `Set= 8`. In the original ordering, *Platter* is the third alternative. Hence the eighth data value in the processed data set has a value of 3. The following DATA step writes out the data after the original order has been restored:

```
data _null_;
   set res3;
   by subj;
   if first.subj then do;
      if mod(subj, 3) eq 1 then put;
      put subj 4. +1 @@;
      end;
   put choose 1. @@;
   run;
```

The results are as follows:

```
   1 334533434233312433   2 334213442433333325   3 333333333333313333
   4 334431444434412453   5 335431434233512423   6 334433434433312433
   7 334433434433322433   8 334433434433412423   9 334433332353312433
  10 325233435233332433  11 334233434433313333  12 334331334433312353
  13 534333334333312323  14 134421444433412423  15 334333435433312335
  16 334433435333315333  17 534333432453312423  18 334435544433412543
  19 334333335433313433  20 331431434233315533  21 334353534433512323
  22 334333452233312523  23 334333332333312433  24 525221444233322423
  25 354333434433312333  26 334435545233312323  27 334353534233352323
  28 334333332333332333  29 334433534335352423  30 334453434533313433
  31 354333334333312433  32 354331332233332423  33 334424432353312325
  34 334433434433312433  35 334551444453412325  36 334234534433312433
  37 334431434433512423  38 354333334433352523  39 334351334333312533
  40 324433334433412323  41 334433444433412443  42 334433434433312423
  43 334434454433332423  44 334433434233312423  45 334451544433412424
  46 434431435433512423  47 524434534433412433  48 335453334433322453
  49 334533434133312433  50 334433332333312423
```

The results match the data on page 325.

The data can be combined with the design and analyzed as in the previous example.

Vacation Example

This example illustrates the design and analysis for a larger choice experiment. We discuss designing a choice experiment, evaluating the design, generating the questionnaire, processing the data, binary coding, generic attributes, quantitative price effects, quadratic price effects, effects coding, alternative-specific effects, analysis, and interpretation of the results. In this example, a researcher is interested in studying choice of vacation destinations. There are five destinations (alternatives) of interest: Hawaii, Alaska, Mexico, California, and Maine. Two summaries of the design are displayed next, one with factors first grouped by attribute and one grouped by destination:

Factor	Destination	Attribute	Levels
X1	Hawaii	Accommodations	Cabin, Bed & Breakfast, Hotel
X2	Alaska	Accommodations	Cabin, Bed & Breakfast, Hotel
X3	Mexico	Accommodations	Cabin, Bed & Breakfast, Hotel
X4	California	Accommodations	Cabin, Bed & Breakfast, Hotel
X5	Maine	Accommodations	Cabin, Bed & Breakfast, Hotel
X6	Hawaii	Scenery	Mountains, Lake, Beach
X7	Alaska	Scenery	Mountains, Lake, Beach
X8	Mexico	Scenery	Mountains, Lake, Beach
X9	California	Scenery	Mountains, Lake, Beach
X10	Maine	Scenery	Mountains, Lake, Beach
X11	Hawaii	Price	$999, $1249, $1499
X12	Alaska	Price	$999, $1249, $1499
X13	Mexico	Price	$999, $1249, $1499
X14	California	Price	$999, $1249, $1499
X15	Maine	Price	$999, $1249, $1499

Factor	Destination	Attribute	Levels
X1	Hawaii	Accommodations	Cabin, Bed & Breakfast, Hotel
X6		Scenery	Mountains, Lake, Beach
X11		Price	$999, $1249, $1499
X2	Alaska	Accommodations	Cabin, Bed & Breakfast, Hotel
X7		Scenery	Mountains, Lake, Beach
X12		Price	$999, $1249, $1499
X3	Mexico	Accommodations	Cabin, Bed & Breakfast, Hotel
X8		Scenery	Mountains, Lake, Beach
X13		Price	$999, $1249, $1499
X4	California	Accommodations	Cabin, Bed & Breakfast, Hotel
X9		Scenery	Mountains, Lake, Beach
X14		Price	$999, $1249, $1499
X5	Maine	Accommodations	Cabin, Bed & Breakfast, Hotel
X10		Scenery	Mountains, Lake, Beach
X15		Price	$999, $1249, $1499

Each alternative is composed of three factors: package cost ($999, $1,249, $1,499), scenery (mountains, lake, beach), and accommodations (cabin, bed & breakfast, and hotel). There are five destinations, each with three attributes, for a total of 15 factors. This problem requires a design with 15 three-level factors, denoted 3^{15}. Each row of the design matrix contains the description of the five alternatives in one choice set. Note that the levels do not have to be the same for all destinations. For example, the cost for Hawaii and Alaska could be different from the other destinations. However, for this example, each destination has the same attributes.

Set Up

We can use the `%MktRuns` autocall macro to suggest design sizes. (All of the autocall macros used in this book are documented starting on page 803.) To use this macro, you specify the number of levels for each of the factors. With 15 attributes each with three prices, you specify fifteen 3's (`3 3 3 3 3 3 3 3 3 3 3 3 3 3 3`), or you can use the more compact syntax of `3 ** 15` as follows:

```
   title 'Vacation Example';

   %mktruns(3 ** 15)
```

The results are as follows:

```
                               Vacation Example

                                Design Summary

                             Number of
                             Levels       Frequency

                                3           15

                               Vacation Example

                     Saturated      = 31
                     Full Factorial = 14,348,907

                     Some Reasonable                   Cannot Be
                        Design Sizes        Violations    Divided By

                                  36                 0
                                  45                 0
                                  54 *               0
                                  63                 0
                                  72 *               0
                                  33                105    9
                                  39                105    9
                                  42                105    9
                                  48                105    9
                                  51                105    9
                                  31 S              120    3 9

                 * - 100% Efficient design can be made with the MktEx macro.
                 S - Saturated Design - The smallest design that can be made.
                     Note that the saturated design is not one of the
                     recommended designs for this problem.  It is shown
                     to provide some context for the recommended sizes.

                                     Vacation Example

                 n    Design                                        Reference

                54    2 **  1  3 ** 25                           Orthogonal Array
                54    2 **  1  3 ** 21   9 **  1                 Orthogonal Array
                54             3 ** 24   6 **  1                 Orthogonal Array
                54             3 ** 20   6 **  1   9 **  1       Orthogonal Array
                54             3 ** 18  18 **  1                 Orthogonal Array
                72    2 ** 23  3 ** 24                           Orthogonal Array
                72    2 ** 22  3 ** 20   6 **  1                 Orthogonal Array
                72    2 ** 21  3 ** 16   6 **  2                 Orthogonal Array
                72    2 ** 20  3 ** 24   4 **  1                 Orthogonal Array
                72    2 ** 19  3 ** 20   4 **  1   6 **  1       Orthogonal Array
                72    2 ** 18  3 ** 16   4 **  1   6 **  2       Orthogonal Array
                72    2 ** 16  3 ** 25                           Orthogonal Array
                72    2 ** 15  3 ** 21   6 **  1                 Orthogonal Array
                72    2 ** 14  3 ** 24   6 **  1                 Orthogonal Array
                72    2 ** 14  3 ** 17   6 **  2                 Orthogonal Array
                72    2 ** 13  3 ** 25   4 **  1                 Orthogonal Array
                72    2 ** 13  3 ** 20   6 **  2                 Orthogonal Array
                72    2 ** 12  3 ** 24  12 **  1                 Orthogonal Array
                72    2 ** 12  3 ** 21   4 **  1   6 **  1       Orthogonal Array
                72    2 ** 12  3 ** 16   6 **  3                 Orthogonal Array
                72    2 ** 11  3 ** 24   4 **  1   6 **  1       Orthogonal Array
                72    2 ** 11  3 ** 20   6 **  1  12 **  1       Orthogonal Array
                72    2 ** 11  3 ** 17   4 **  1   6 **  2       Orthogonal Array
                72    2 ** 10  3 ** 20   4 **  1   6 **  2       Orthogonal Array
                72    2 ** 10  3 ** 16   6 **  2  12 **  1       Orthogonal Array
                72    2 **  9  3 ** 16   4 **  1   6 **  3       Orthogonal Array
                72             3 ** 25   8 **  1                 Orthogonal Array
                72             3 ** 24  24 **  1                 Orthogonal Array
```

The output tells us the size of the saturated design, which is the number of parameters in the linear arrangement, and suggests design sizes.

In this design, there are $15 \times (3 - 1) + 1 = 31$ parameters, so at least 31 choice sets must be created. With all three-level factors, the number of choice sets in all orthogonal and balanced designs must be divisible by $3 \times 3 = 9$. Hence, optimal designs for this problem have at least 36 choice sets (the smallest number ≥ 31 and divisible by 9). Note, however, that zero violations does not guarantee that a 100% *D*-efficient design exists. It just means that 100% *D*-efficiency is not precluded by unequal cell frequencies. In fact, the `%MktEx` orthogonal design catalog does not include orthogonal designs for this problem in 36, 45, and 63 runs (because they do not exist).

Thirty-six is a good design size (2 blocks of size 18) as is 54 (3 blocks of size 18). Fifty-four is probably the best choice, and that is what we recommend for this study. However, we instead create a *D*-efficient experimental design with 36 choice sets using the `%MktEx` macro. In practice, with more difficult designs, an orthogonal design is not available, and using 36 choice sets lets us see an example of using the `%Mkt` family of macros to get a nonorthogonal design.

We can see what orthogonal designs with three-level factors are available in 36 runs as follows. The `%MktOrth` macro creates a data set with information about the orthogonal designs that the `%MktEx` macro knows how to make. This macro produces a data set called `MktDesLev` that contains variables `n`, the number of runs; `Design`, a description of the design; and `Reference`, which contains the type of the design. In addition, there are variables: `x1`, the number of 1-level factors (which is always zero); `x2`, the number of 2-level factors; `x3`, the number of 3-level factors; and so on. The following steps use `%MktOrth` to only output `n=36` run designs and sort this list so that designs with the most three-level factors are displayed first:

```
%mktorth(range=n=36)

proc sort data=mktdeslev out=list(drop=x:);
   by descending x3;
   where x3;
   run;

proc print; run;
```

The results are as follows:

```
                              Vacation Example

               Obs     n    Design                            Reference

                 1    36    2 **  4   3 ** 13                 Orthogonal Array
                 2    36              3 ** 13    4 **  1      Orthogonal Array
                 3    36    2 ** 11   3 ** 12                 Orthogonal Array
                 4    36    2 **  2   3 ** 12    6 **  1      Orthogonal Array
                 5    36              3 ** 12   12 **  1      Orthogonal Array
                 6    36    2 **  3   3 **  9    6 **  1      Orthogonal Array
                 7    36    2 ** 10   3 **  8    6 **  1      Orthogonal Array
                 8    36    2 **  1   3 **  8    6 **  2      Orthogonal Array
                 9    36              3 **  7    6 **  3      Orthogonal Array
                10    36    2 **  2   3 **  5    6 **  2      Orthogonal Array
```

```
          11    36    2 ** 16  3 **  4               Orthogonal Array
          12    36    2 **  9  3 **  4   6 **  2     Orthogonal Array
          13    36    2 **  1  3 **  3   6 **  3     Orthogonal Array
          14    36    2 ** 20  3 **  2               Orthogonal Array
          15    36    2 ** 13  3 **  2   6 **  1     Orthogonal Array
          16    36    2 **  3  3 **  2   6 **  3     Orthogonal Array
          17    36    2 ** 27  3 **  1               Orthogonal Array
          18    36    2 ** 18  3 **  1   6 **  1     Orthogonal Array
          19    36    2 ** 10  3 **  1   6 **  2     Orthogonal Array
          20    36    2 **  4  3 **  1   6 **  3     Orthogonal Array
```

There are 13 two-level factors available in 36 runs, and we need 15, only two more, so we expect to make a pretty good nonorthogonal design.

Designing the Choice Experiment

The following step creates a design:

```
%let m   = 6;                      /* m alts including constant        */
%let mm1 = %eval(&m - 1);          /* m - 1                            */
%let n   = 18;                     /* number of choice sets per person */
%let blocks = 2;                   /* number of blocks                 */

%mktex(3 ** 15 2, n=&n * &blocks, seed=151)
```

The specification `3 ** 15` requests a design with 15 factors, `x1---x15`, each with three levels. This specification also requests a two-level factor (the `2` following the `3 ** 15`). This is because 36 choice sets might be too many for one person to rate, so we might want to block the design into two blocks, and we can use a two-level factor to do this. A design with $18 \times 2 = 36$ runs is requested, which means 36 choice sets. A random number seed is explicitly specified so we can reproduce these exact results.[*]

Some of the log messages are as follows:

```
NOTE: Generating the candidate set.
NOTE: Performing 20 searches of 81 candidates, full-factorial=28,697,814.
NOTE: Generating the orthogonal array design, n=36.
```

The macro searches a fractional-factorial candidate set of 81 runs, and it also generates an orthogonal array in 36 runs to try as part of the design. This is explained in more detail on page 347.

[*]By specifying a random number seed, results should be reproducible within a SAS release for a particular operating system and for a particular version of the macro. However, due to machine and macro differences, some results might not be exactly reproducible everywhere. For most orthogonal and balanced designs, the results should be reproducible. When computerized searches are done, it is likely that you will not get the same design as the one in the book, although you expect the efficiency differences to be slight.

Some of the results from the `%MktEx` macro are as follows:

```
                              Vacation Example

                          Algorithm Search History

                              Current          Best
Design    Row,Col  D-Efficiency  D-Efficiency  Notes
----------------------------------------------------------
     1      Start       82.2544       82.2544  Can
     1        End       82.2544

     2      Start       78.8337                Tab,Ran
     2      3  15       82.7741       82.7741
     2      4  14       83.2440       83.2440
     2      4  15       83.6041       83.6041
     2      4  15       83.8085       83.8085
     2      6  14       84.0072       84.0072
     .
     .
     .
     2        End       98.8567
     .
     .
     .

     5      Start       78.5222                Tab,Ran
     5     11  14       98.8567       98.8567
     5        End       98.8567
     .
     .
     .

     8      Start       77.5829                Tab,Ran
     8     10  15       98.9438       98.9438
     8        End       98.9438
     .
     .
     .

    21      Start       48.8411                Ran,Mut,Ann
    21        End       93.1010
                           Design Search History

                              Current          Best
Design    Row,Col  D-Efficiency  D-Efficiency  Notes
----------------------------------------------------------
     0    Initial       98.9438       98.9438  Ini
```

```
       1      Start       77.8094                 Tab,Ran
       1        End       98.6368

       2      Start       77.9170                 Tab,Ran
       2        End       98.5516
       .
       .
       .
      78      Start       79.9023                 Tab,Ran
      78     24  15       98.9438      98.9438
      78        End       98.9438
       .
       .
       .
      87      Start       74.7014                 Tab,Ran
      87      4  15       98.9438      98.9438
      87        End       98.9438
       .
       .
       .
     146      Start       78.3794                 Tab,Ran
     146     19  15       98.9438      98.9438
     146        End       98.9438
       .
       .
       .
     200      Start       84.1995                 Tab,Ran
     200        End       98.6368

                         Vacation Example

                     Design Refinement History

                        Current         Best
Design    Row,Col  D-Efficiency  D-Efficiency  Notes
----------------------------------------------------------
     0    Initial       98.9438       98.9438  Ini

     1      Start       96.5678                Pre,Mut,Ann
     1      1   8       98.9438       98.9438
     1     26  14       98.9438       98.9438
     1     30  11       98.9438       98.9438
     1      1  12       98.9438       98.9438
     1     32   5       98.9438       98.9438
     1     18   6       98.9438       98.9438
     1        End       98.9438
```

```
                 .
                 .
                 .

                 6      Start       97.2440                Pre,Mut,Ann
                 6     33   7       98.9438       98.9438
                 6      4   3       98.9438       98.9438
                 6     16  12       98.9438       98.9438
                 6      3  14       98.9438       98.9438
                 6     20  15       98.9438       98.9438
                 6        End       98.6958

NOTE: Stopping since it appears that no improvement is possible.

                              Vacation Example

                            The OPTEX Procedure

                          Class Level Information

                           Class  Levels  Values

                           x1       3     1 2 3
                           x2       3     1 2 3
                           x3       3     1 2 3
                           x4       3     1 2 3
                           x5       3     1 2 3
                           x6       3     1 2 3
                           x7       3     1 2 3
                           x8       3     1 2 3
                           x9       3     1 2 3
                           x10      3     1 2 3
                           x11      3     1 2 3
                           x12      3     1 2 3
                           x13      3     1 2 3
                           x14      3     1 2 3
                           x15      3     1 2 3
                           x16      2     1 2

                              Vacation Example
                                                                 Average
                                                                Prediction
    Design                                                       Standard
    Number     D-Efficiency     A-Efficiency     G-Efficiency      Error
    ------------------------------------------------------------------------
         1        98.9437          97.9592          98.9743          0.9428
```

The `%MktEx` macro used 30 seconds and found a design that is almost 99% *D*-efficient. (Differences in the fourth decimal place between the iteration history and the final table, in this case 98.9438 versus 98.9437, are due to rounding error and differences in ridging strategies between the macro code the generates the design and PROC OPTEX, which evaluates the design, and are nothing to worry about.)

The %MktEx Macro Algorithm

The `%MktEx` macro creates *D*-efficient linear experimental designs using several approaches. The macro tries to create an orthogonal array, it searches a set of candidate runs (rows of the design), and it uses a coordinate-exchange algorithm using both a random initial designs and also a partial orthogonal array initialization. The macro stops if at any time it finds a perfect, 100% *D*-efficient, orthogonal and balanced design. This first phase is the algorithm search phase. In it, the macro tries a number of methods for this problem. At the end of this phase, the macro chooses the method that has produced the best design and performs another set of iterations using exclusively the chosen approach. Finally, the macro performs a third set of iterations, where it takes the best design it found so far and tries to improve it.

The `%MktEx` macro can directly generate, without iterations, well over one-hundred thousand different 100% *D*-efficient, orthogonal and balanced designs. It does this using its design catalog and many different general and ad hoc algorithms. The closest design that the macro knows how to make for this problem is $2^1 3^{13}$ in 36 runs.

The candidate-set search has two parts. First, either PROC PLAN is run to create a full-factorial design for small problems, or PROC FACTEX is run to create a fractional-factorial design for large problems. Either way, this larger design is a *candidate set* that in the second part is searched by PROC OPTEX using the modified Fedorov algorithm. A design is built from a selection of the rows of the candidate set (Fedorov 1972; Cook and Nachtsheim 1980). The modified Fedorov algorithm considers each run in the design and each candidate run. Candidate runs are swapped in and design runs are swapped out if the swap improves *D*-efficiency. In this case, since the full-factorial design is large (over 14 million runs), the candidate-set search step calls PROC FACTEX to make the candidate set and then PROC OPTEX to do the search. The `Can` line of the iteration history shows that this step found a design that is 82.2544% *D*-efficient.

Next, the `%MktEx` macro uses the *coordinate-exchange algorithm*, based on Meyer and Nachtsheim (1995). The coordinate-exchange algorithm considers each level of each factor, and considers the effect on *D*-efficiency of changing a level ($1 \rightarrow 2$, or $1 \rightarrow 3$, or $2 \rightarrow 1$, or $2 \rightarrow 3$, or $3 \rightarrow 1$, or $3 \rightarrow 2$, and so on). Exchanges that increase *D*-efficiency are performed. In this step, the macro first tries to initialize the design with an orthogonal array (designated by `Tab`, which refers to the orthogonal array table or catalog) and a random design (`Ran`) both. In this case, 14 of the 16 factors can be initialized with the 13 three-level factors and one two-level factor of $2^4 3^{13}$, and the other two factors are randomly initialized. Levels that are not orthogonally initialized might be exchanged for other levels if the exchange increases *D*-efficiency. The algorithm search and design search iteration histories for this example show that the macro exchanged levels in factor 14 and 15 only, the ones that are randomly initialized.

The initialization might be more complicated in other problems. Say you asked for the design $4^1 5^1 3^4$ in 18 runs. The macro uses the orthogonal array $3^6 6^1$ in 18 runs to initialize the three-level factors orthogonally, and the five-level factor with the six-level factor coded down to five levels (which is unbalanced). The four-level factor is randomly initialized. The macro also tries the same initialization but with a random rather than unbalanced initialization of the five-level factor, as a minor variation on the first initialization. In the next initialization variation, the macro uses a fully-random initialization. If the number of runs requested is smaller than the number or runs in the initial orthogonal array, the macro initializes the design with just the first n rows of the orthogonal array. Similarly, if the number of runs requested is larger than the number or runs in the initial orthogonal array, the macro initializes part of the design with the orthogonal orthogonal array and the remaining rows and columns randomly. The coordinate-exchange algorithm considers each level of each factor that is not orthogonally initialized, and it exchanges a level if the exchange improves D-efficiency. When the number or runs in the orthogonal array does not match the number of runs desired, none of the design is initialized orthogonally.

The coordinate-exchange algorithm is not restricted by having a candidate set and hence can *potentially* consider any possible design. In practice, however, both the candidate-set-based and coordinate-exchange algorithms consider only a tiny fraction of the possible designs. When the number of runs in the full-factorial design is very small (say 100 or 200 runs), the modified Fedorov algorithm and coordinate-exchange algorithms usually work equally well. When the number of runs in the full-factorial design is small (up to several thousand), the modified Fedorov algorithm is sometimes superior to coordinate exchange, particularly for models with interactions. When the full-factorial design is larger, coordinate exchange is usually the superior approach. However, heuristics like these are sometimes wrong, which is why the macro tries both methods to see which one is really best for each problem.

In the first attempt at coordinate exchange (Design 2), the macro found a design that is 98.8567% D-efficient (Design 2, `End`). In design 3 and subsequent designs, the macro uses this same approach, but different random initializations of the remaining two factors. In design 8, the `%MktEx` macro finds a design that is 98.9438% D-efficient. Designs 12 through 21 use a purely random initialization and simulated annealing and are not as good as previous designs. During these iterations, the macro is considering exchanging every level of every factor with every other level, one one row and one factor at a time. At this point, the `%MktEx` macro determines that the combination of orthogonal array and random initialization is working best and tries more iterations using that approach. It starts by displaying the initial (`Ini`) best D-efficiency of 98.9438. In designs 78, 87, 146, and 197 the macro finds a design that is 98.9438% D-efficient.

Next, the `%MktEx` macro tries to improve the best design it found previously. Using the previous best design as an initialization (`Pre`), and random mutations of the initialization (`Mut`) and simulated annealing (`Ann`), the macro uses the coordinate-exchange algorithm to try to find a better design. This step is important because the best design that the macro found might be an intermediate design, and it might not be the final design at the end of an iteration. Sometimes the iterations deliberately make the designs less D-efficient, and sometimes, the macro never finds a design as efficient or more efficient again. Hence it is worthwhile to see if the best design found so far can be improved. In this case, the macro fails to improve the design. After iteration 6, the macro stops since it keeps finding the same design over and over. This does not necessarily mean the macro found *the* optimal design; it means it found a very attractive (perhaps local) optimum, and it is unlikely it will do better using this approach. At the end, PROC OPTEX is called to display the levels of each factor and the final D-efficiency.

Random mutations add random noise to the initial design before the iterations start (levels are randomly changed). This might eliminate the perfect balance that is often in the initial design. By default, random mutations are used with designs with fully-random initializations and in the design refinement step; orthogonal initial designs are not mutated.

Simulated annealing allows the design to get worse occasionally but with decreasing probability as the number of exchanges increases. For design 1, for the first level of the first factor, by default, the macro might execute an exchange (say change a 2 to a 1), that makes the design worse, with probability

0.05. As more and more exchanges occur, this probability decreases so at the end of the processing of design 1, exchanges that decrease *D*-efficiency are hardly ever done. For design 2, this same process is repeated, again starting by default with an annealing probability of 0.05. This often helps the algorithm overcome local efficiency maxima. To envision this, imagine that you are standing on a molehill next to a mountain. The only way you can start going up the mountain is to first step down off the molehill. Once you are on the mountain, you might occasionally hit a dead end, where all you can do is step down and look for a better place to continue going up. Other analogies include cleaning a garage and painting a room. Both have steps where you make things look worse so that in the end they look better. The solitaire game "Spider," which is available on many PCs, is another example. Sometimes, you need to temporarily break apart those suits that you so carefully put together in order to make progress. Simulated annealing, by occasionally stepping down the efficiency function, often allows the macro to go farther up it than it would otherwise. The simulated annealing is why you sometimes see designs getting worse in the iteration history. However, the macro keeps track of the best design, not the final design in each step. By default, annealing is used with designs with fully-random initializations and in the design refinement step; simulated annealing is not used with orthogonal initial designs.

For this example, the `%MktEx` macro ran in around 30 seconds. If an orthogonal design had been available, run time would have been a few seconds. If the fully-random initialization method had been the best method, run time might have been on the order of 10 to 45 minutes. Since the orthogonal array initialization worked best, run time is much shorter. While it is possible to construct huge problems that take much longer, for any design that most marketing researchers are likely to encounter, run time should be less than one hour. One of the macro options, `maxtime=`, typically ensures this.

Examining the Design

Before you use a design, you should always look at its characteristics. We can use the `%MktEval` macro to do this as follows:

```
   %mkteval(data=randomized)
```

The results are as follows:

```
        x1  x2  x3  x4  x5  x6  x7  x8  x9  x10 x11 x12 x13  x14  x15  x16

    x1  1   0   0   0   0   0   0   0   0   0   0   0   0    0    0    0
    x2  0   1   0   0   0   0   0   0   0   0   0   0   0    0    0    0
    x3  0   0   1   0   0   0   0   0   0   0   0   0   0    0    0    0
    x4  0   0   0   1   0   0   0   0   0   0   0   0   0    0    0    0
    x5  0   0   0   0   1   0   0   0   0   0   0   0   0    0    0    0
    x6  0   0   0   0   0   1   0   0   0   0   0   0   0    0    0    0
    x7  0   0   0   0   0   0   1   0   0   0   0   0   0    0    0    0
    x8  0   0   0   0   0   0   0   1   0   0   0   0   0    0    0    0
    x9  0   0   0   0   0   0   0   0   1   0   0   0   0    0    0    0
    x10 0   0   0   0   0   0   0   0   0   1   0   0   0    0    0    0
    x11 0   0   0   0   0   0   0   0   0   0   1   0   0    0    0    0
    x12 0   0   0   0   0   0   0   0   0   0   0   1   0    0    0    0
    x13 0   0   0   0   0   0   0   0   0   0   0   0   1    0.25 0.25 0
    x14 0   0   0   0   0   0   0   0   0   0   0   0   0.25 1    0.25 0
    x15 0   0   0   0   0   0   0   0   0   0   0   0   0.25 0.25 1    0
    x16 0   0   0   0   0   0   0   0   0   0   0   0   0    0    0    1

                          Vacation Example
                       Summary of Frequencies
            There are 0 Canonical Correlations Greater Than 0.316
                     * - Indicates Unequal Frequencies

                       Frequencies

           x1          12 12 12
           x2          12 12 12
           x3          12 12 12
           x4          12 12 12
           x5          12 12 12
           x6          12 12 12
           x7          12 12 12
           x8          12 12 12
           x9          12 12 12
           x10         12 12 12
           x11         12 12 12
           x12         12 12 12
           x13         12 12 12
           x14         12 12 12
           x15         12 12 12
           x16         18 18
```

```
          x1 x2      4 4 4 4 4 4 4 4 4
          x1 x3      4 4 4 4 4 4 4 4 4
          x1 x4      4 4 4 4 4 4 4 4 4
          x1 x5      4 4 4 4 4 4 4 4 4
          x1 x6      4 4 4 4 4 4 4 4 4
          x1 x7      4 4 4 4 4 4 4 4 4
          x1 x8      4 4 4 4 4 4 4 4 4
          x1 x9      4 4 4 4 4 4 4 4 4
          x1 x10     4 4 4 4 4 4 4 4 4
          x1 x11     4 4 4 4 4 4 4 4 4
          x1 x12     4 4 4 4 4 4 4 4 4
          x1 x13     4 4 4 4 4 4 4 4 4
          x1 x14     4 4 4 4 4 4 4 4 4
          x1 x15     4 4 4 4 4 4 4 4 4
          x1 x16     6 6 6 6 6 6
          .
          .
          .
     *    x13 x14    3 6 3 6 3 3 3 3 6
     *    x13 x15    3 3 6 3 6 3 6 3 3
          x13 x16    6 6 6 6 6 6
     *    x14 x15    3 6 3 3 3 6 6 3 3
          x14 x16    6 6 6 6 6 6
          x15 x16    6 6 6 6 6 6
          N-Way      1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

This design looks great! The factors `x1-x13` form an orthogonal design, `x14` and `x15` are slightly correlated with each other and with `x13`. The blocking factor `x16` is orthogonal to all the other factors. All of the factors are perfectly balanced. The N-Way frequencies show that each choice set appears once.

What if there had been some larger canonical correlations? Would this be a problem? That depends. You have to decide this for yourself based on your particular study. You do not want large correlations between your most important factors. If you have high correlations between the wrong factors, you can swap them with other factors with the same number of levels, or try to make a new design with a different seed, or change the number of choice sets, and so on. While this design looks great, we should make one minor adjustment based on these results. Since our correlations are in the factors we originally planned to make price factors, we should change our plans slightly and use those factors for less important attributes like scenery.

You can run the `%MktEx` macro to provide additional information about a design, for example, asking to examine the information matrix (`I`) and its inverse (`V`), which is the variance matrix of the parameter estimates. You hope to see that all of the off-diagonal elements of the variance matrix, the covariances, are small relative to the variances on the diagonal. When `options=check` is specified, the macro evaluates an initial design instead of generating a design. The option `init=randomized` names the design to evaluate, and the `examine=` option displays the information and variance matrices. The blocking variable is dropped.

The following step creates the design:

```
%mktex(3 ** 15,                   /* all attrs of all alternatives       */
       n=&n * &blocks,            /* total number of choice sets         */
       init=randomized(drop=x16), /* initial design                      */
       options=check,             /* check initial design efficiency     */
       examine=i v)               /* show information & variance matrices */
```

A small part of the output is as follows:

```
                                 Vacation Example

                                                                     Average
                                                                    Prediction
     Design                                                          Standard
     Number     D-Efficiency     A-Efficiency     G-Efficiency         Error
     ------------------------------------------------------------------------
          1          98.9099          97.8947          98.9418          0.9280

                                  Vacation Example
                                 Information Matrix

            Intercept x11     x12     x21     x22     x31     x32     x41

  Intercept   36.000   0       0       0       0       0       0       0
  x11          0      36.000   0       0       0       0       0       0
  x12          0       0      36.000   0       0       0       0       0
  x21          0       0       0      36.000   0       0       0       0
  x22          0       0       0       0      36.000   0       0       0
  x31          0       0       0       0       0      36.000   0       0
  x32          0       0       0       0       0       0      36.000   0
  x41          0       0       0       0       0       0       0      36.000
 .
 .
 .

  x122       36.000    0         0         0         0         0         0
  x131         0      36.000     0         9.000     0        -4.500    -7.794
  x132         0       0        36.000     0         9.000    -7.794     4.500
  x141         0       9.000     0        36.000     0        -4.500    -7.794
  x142         0       0         9.000     0        36.000    -7.794     4.500
  x151         0      -4.500    -7.794    -4.500    -7.794    36.000     0
  x152         0      -7.794     4.500    -7.794     4.500     0        36.000
```

```
                    Vacation Example
                    Variance Matrix

          Intercept x11     x12     x21     x22     x31     x32     x41

 Intercept    0.028    0       0       0       0       0       0       0
 x11          0        0.028   0       0       0       0       0       0
 x12          0        0       0.028   0       0       0       0       0
 x21          0        0       0       0.028   0       0       0       0
 x22          0        0       0       0       0.028   0       0       0
 x31          0        0       0       0       0       0.028   0       0
 x32          0        0       0       0       0       0       0.028   0
 x41          0        0       0       0       0       0       0       0.028
.
.
.

 x122        0.028     0         0         0         0         0         0
 x131        0         0.031     0        -0.006     0         0.003     0.005
 x132        0         0         0.031     0        -0.006     0.005    -0.003
 x141        0        -0.006     0         0.031     0         0.003     0.005
 x142        0         0        -0.006     0         0.031     0.005    -0.003
 x151        0         0.003     0.005     0.003     0.005     0.031     0
 x152        0         0.005    -0.003     0.005    -0.003     0         0.031
```

This design still looks good. The *D*-efficiency for the design excluding the blocking factor is 98.9099%. We can see that the nonorthogonality between `x13-x15` makes their variances larger than the other factors (0.031 versus 0.028).

These next steps use the `%MktLab` macro to reassign the variable names, store the design in a permanent SAS data set, `sasuser.VacationLinDesBlckd`, and then use the `%MktEx` macro to check the results. See page 309 for more information about permanent SAS data sets. We need to make the correlated variables correspond to the least important attributes in different alternatives (in this case the scenery factors for Alaska, Mexico, and Maine). The `vars=` option provides the new variable names: the first variable (originally `x1`) becomes `x1` (still), ..., the fifth variable (originally `x5`) becomes `x5` (still), the sixth variable (originally `x6`) becomes `x11`, ... the tenth variable (originally `x10`) becomes `x15`, the eleventh through fifteenth original variables become `x6`, `x9`, `x7`, `x8`, `x10`, and finally the last variable becomes `Block`. The following PROC SORT step sorts the design into blocks:

```
%mktlab(data=randomized, vars=x1-x5 x11-x15 x6 x9 x7 x8 x10 Block,
        out=sasuser.VacationLinDesBlckd)

proc sort data=sasuser.VacationLinDesBlckd; by block; run;

%mkteval(blocks=block)
```

The output from the `%MktLab` macro, which shows the correspondence between the original and new variable names is as follows:

```
      Variable Mapping:
         x1  : x1
         x2  : x2
         x3  : x3
         x4  : x4
         x5  : x5
         x6  : x11
         x7  : x12
         x8  : x13
         x9  : x14
         x10 : x15
         x11 : x6
         x12 : x9
         x13 : x7
         x14 : x8
         x15 : x10
         x16 : Block
```

Some of the output from the `%MktEval` macro is as follows:

```
                              Vacation Example
                   Canonical Correlations Between the Factors
             There are 0 Canonical Correlations Greater Than 0.316

          Block x1  x2  x3  x4  x5  x11 x12 x13 x14 x15 x6  x9  x7   x8   x10

    Block 1     0   0   0   0   0   0   0   0   0   0   0   0   0    0    0
    x1    0     1   0   0   0   0   0   0   0   0   0   0   0   0    0    0
    x2    0     0   1   0   0   0   0   0   0   0   0   0   0   0    0    0
    x3    0     0   0   1   0   0   0   0   0   0   0   0   0   0    0    0
    x4    0     0   0   0   1   0   0   0   0   0   0   0   0   0    0    0
    x5    0     0   0   0   0   1   0   0   0   0   0   0   0   0    0    0
    x11   0     0   0   0   0   0   1   0   0   0   0   0   0   0    0    0
    x12   0     0   0   0   0   0   0   1   0   0   0   0   0   0    0    0
    x13   0     0   0   0   0   0   0   0   1   0   0   0   0   0    0    0
    x14   0     0   0   0   0   0   0   0   0   1   0   0   0   0    0    0
    x15   0     0   0   0   0   0   0   0   0   0   1   0   0   0    0    0
    x6    0     0   0   0   0   0   0   0   0   0   0   1   0   0    0    0
    x9    0     0   0   0   0   0   0   0   0   0   0   0   1   0    0    0
    x7    0     0   0   0   0   0   0   0   0   0   0   0   0   1    0.25 0.25
    x8    0     0   0   0   0   0   0   0   0   0   0   0   0   0.25 1    0.25
    x10   0     0   0   0   0   0   0   0   0   0   0   0   0   0.25 0.25 1
```

```
                      Vacation Example
                   Summary of Frequencies
        There are 0 Canonical Correlations Greater Than 0.316
              * - Indicates Unequal Frequencies

                         Frequencies

            Block        18 18
            x1           12 12 12
            x2           12 12 12
            x3           12 12 12
            x4           12 12 12
            x5           12 12 12
            x11          12 12 12
            x12          12 12 12
            x13          12 12 12
            x14          12 12 12
            x15          12 12 12
            x6           12 12 12
            x9           12 12 12
            x7           12 12 12
            x8           12 12 12
            x10          12 12 12
            Block x1     6 6 6 6 6 6
            Block x2     6 6 6 6 6 6
            Block x3     6 6 6 6 6 6
            Block x4     6 6 6 6 6 6
            Block x5     6 6 6 6 6 6
            Block x11    6 6 6 6 6 6
            Block x12    6 6 6 6 6 6
            Block x13    6 6 6 6 6 6
            Block x14    6 6 6 6 6 6
            Block x15    6 6 6 6 6 6
            Block x6     6 6 6 6 6 6
            Block x9     6 6 6 6 6 6
            Block x7     6 6 6 6 6 6
            Block x8     6 6 6 6 6 6
            Block x10    6 6 6 6 6 6
            x1 x2        4 4 4 4 4 4 4 4 4
            x1 x3        4 4 4 4 4 4 4 4 4
            x1 x4        4 4 4 4 4 4 4 4 4
            x1 x5        4 4 4 4 4 4 4 4 4
            .
            .
            .
```

```
*    x7 x8        6 3 3 3 6 3 3 3 6
*    x7 x10       3 3 6 3 6 3 6 3 3
*    x8 x10       3 3 6 3 6 3 6 3 3
     N-Way        1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

From a Linear Arrangement to a Choice Design

These next steps prepare the design for analysis and further evaluation. We need to covert our linear arrangement into a choice design.* We need to create a data set Key that describes how the factors in our linear arrangement are used to make the choice design for analysis. The Key data set contains all of the factor names, x1, x2, x3, ... x15. We can run the %MktKey macro to get these names, for cutting and pasting into the program without typing them. The following step requests 5 rows, 3 columns and the results transposed so names progress down each column instead of across each row:

```
%mktkey(5 3 t)
```

The %MktKey macro produces the following data set:

```
x1    x2     x3

x1    x6     x11
x2    x7     x12
x3    x8     x13
x4    x9     x14
x5    x10    x15
```

*See page 67 for an explanation of the linear arrangement of a choice design versus the arrangement of a choice design that is more suitable for analysis.

The following steps make the Key data set and processes the design:

```
title 'Vacation Example';

data key;
   input Place $ 1-10 (Lodge Scene Price) ($);
   datalines;
Hawaii      x1    x6     x11
Alaska      x2    x7     x12
Mexico      x3    x8     x13
California  x4    x9     x14
Maine       x5    x10    x15
Home        .     .      .
;

%mktroll(design=sasuser.VacationLinDesBlckd, key=key, alt=place,
         out=sasuser.VacationChDes)
```

For analysis, the design has four factors as shown by the variables in the data set Key. Place is the alternative name; its values are directly read from the Key in-stream data. Lodge is an attribute whose values are constructed from the sasuser.VacationLinDesBlckd data set. Lodge is created from x1 for Hawaii, x2 for Alaska, ..., x5 for Maine, and no attribute for Home. Similarly, Scene is created from x6-x10, and Price is created from x11-x15. The macro %MktRoll is used to create the data set sasuser.VacationChDes from sasuser.VacationLinDesBlckd using the mapping in Key and using the variable Place as the alternative ID variable.

The macro displays the following warning:

```
WARNING: The variable BLOCK is in the DESIGN= data set but not
         the KEY= data set.
```

While this message could indicate a problem, in this case it does not. The variable Block in the design=sasuser.VacationLinDesBlckd data set does not appear in the final design. The purpose of the variable Block (sorting the design into blocks) has already been achieved. You can specify options=nowarn if you want to suppress this warning.

These next steps show the results for the first two choice sets:

```
proc print data=sasuser.VacationLinDesBlckd(obs=2);
   id Block;
   var x1-x15;
   run;

proc print data=sasuser.VacationChDes(obs=12);
   id set; by set;
   run;
```

The data set is converted from a design matrix with one row per choice set to a design matrix with one row per alternative per choice set.

The results are as follows:

```
                              Vacation Example

 Block  x1  x2  x3  x4  x5  x6  x7  x8  x9  x10  x11  x12  x13  x14  x15

   1     1   3   3   1   1   3   1   3   2   1    1    1    2    2    2
   1     1   2   1   2   3   3   2   2   1   3    1    3    1    1    2
                              Vacation Example

                    Set    Place         Lodge    Scene    Price

                     1     Hawaii          1        3        1
                           Alaska          3        1        1
                           Mexico          3        3        2
                           California      1        2        2
                           Maine           1        1        2
                           Home            .        .        .
                     2     Hawaii          1        3        1
                           Alaska          2        2        3
                           Mexico          1        2        1
                           California      2        1        1
                           Maine           3        3        2
                           Home            .        .        .
```

The following steps assign formats, convert the variable `Price` to contain actual prices, and recode the constant alternative:

```
proc format;
   value price 1 = ' 999'      2 = '1249'
               3 = '1499'      0 = '   0';
   value scene 1 = 'Mountains' 2 = 'Lake'
               3 = 'Beach'     0 = 'Home';
   value lodge 1 = 'Cabin'     2 = 'Bed & Breakfast'
               3 = 'Hotel'     0 = 'Home';
   run;

data sasuser.VacationChDes;
   set sasuser.VacationChDes;
   if place = 'Home' then do; lodge = 0; scene = 0; price = 0; end;
   price = input(put(price, price.), 5.);
   format scene scene. lodge lodge.;
   run;


proc print data=sasuser.VacationChDes(obs=12);
   id set; by set;
   run;
```

The results are as follows:

```
                          Vacation Example

Set    Place         Lodge               Scene        Price

 1     Hawaii        Cabin               Beach          999
       Alaska        Hotel               Mountains      999
       Mexico        Hotel               Beach         1249
       California    Cabin               Lake          1249
       Maine         Cabin               Mountains     1249
       Home          Home                Home             0
 2     Hawaii        Cabin               Beach          999
       Alaska        Bed & Breakfast     Lake          1499
       Mexico        Cabin               Lake           999
       California    Bed & Breakfast     Mountains      999
       Maine         Hotel               Beach         1249
       Home          Home                Home             0
```

It is not necessary to recode the missing values for the constant alternative. In practice, we usually do not do this step. However, for this first analysis, we want all nonmissing values of the attributes so we can see all levels in the final displayed output. We also recode `Price` so that for a later analysis, we can analyze `Price` as a quantitative effect. For example, the expression `put(price, price.)` converts a number, say 2, into a string (in this case '1249'), then the `input` function reads the string and converts it to a numeric 1249.

Testing the Design Before Data Collection

Collecting data is time consuming and expensive. It is always good practice to make sure that the design works with the most complicated model that we anticipate fitting. The following steps evaluate the choice design:

```
title2 'Evaluate the Choice Design';

%choiceff(data=sasuser.VacationChDes,/* candidate set of choice sets       */
          init=sasuser.VacationChDes(keep=set), /* select these sets       */
          intiter=0,                /* evaluate without internal iterations */
                                    /* alternative-specific effects model   */
                                    /* zero=none - use all levels of place  */
                                    /* order=data - do not sort levels      */
          model=class(place / zero=none order=data)
                                    /* place * price ... - interactions or  */
                                    /*    alternative-specific effects      */
                class(place * price place * scene place * lodge /
                      zero=none     /* zero=none - use all levels of place  */
                      order=formatted) / /* order=formatted - sort levels   */
                lprefix=0           /* lpr=0 labels created from just levels*/
                cprefix=0           /* cpr=0 names created from just levels */
                separators=' ' ', ',/* use comma sep to build interact terms*/

          nsets=36,                 /* number of choice sets                */
          nalts=6,                  /* number of alternatives               */
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

The `%ChoicEff` macro has two uses. You can use it to search for an efficient choice design, or you can use it to evaluate a choice design including designs that are generated using other methods such as the `%MktEx` macro. It is this latter use that is illustrated here.

The way you check a design like this is to first name it in the `data=` option. This is the candidate set that contains all of the choice sets that we will consider. In addition, the same design is named in the `init=` option. Just the variable `Set` is kept. It is used to bring in just the indicated choice sets from the `data=` design, which in this case is all of them. The option `nsets=` specifies that there are 36 choice sets, and `nalts=` specifies that there are 6 alternatives. The option `beta=zero` specifies that we are assuming for design evaluation purposes the null hypothesis that all of the betas or part-worth utilities are zero. You can evaluate the design for other parameter vectors by specifying a list of numbers after `beta=`. This will change the variances and standard errors. We also specify `intiter=0` which specifies zero internal iterations. We use zero internal iterations when we want to evaluate an initial design, but not attempt to improve it. The last option specifies the model.

The model specification contains everything that appears in the TRANSREG procedure's `model` statement for coding the design. Some of these options are familiar from the previous example. The specification `class(place / zero=none order=data)` names the `place` variable as a classification variables and asks for coded variables for every level including the constant, stay-at-home alternative. The specification `class(place * price place * scene place * lodge / zero=none order=formatted)` asks for alternative-specific effects for price, lodging, and scenery. The alternative-specific effects permit the part-worth utilities to be different for each of the destinations. This is accomplished by requesting interactions between the destination and the attributes. `Class` levels are sorted by their formatted values, and for all factors. The `zero=none` option is used so that for now we can see all of the levels of all of the factors. Many of these will not correspond to estimable parameters, and we can eliminate them later. See page 78 for more information about the `zero=` option.

The `lprefix=0` option specifies that when labels are created for the binary variables, zero characters of the original variable name should be used as a prefix. The `cprefix=0` option specifies that when names are created for the binary variables, zero characters of the original variable name should be used as a prefix. The `separators=' ' ', '` option provides two strings (one blank and the other a comma followed by a blank) that let you specify label component separators for the main effect and interaction terms. By specifying a comma and a blank for the second value, we request labels for the side trip effects like `'Alaska, 999'` instead of the default `'Alaska * 999'`. This option is explained in more detail on page 449.

The last tables from the `%ChoicEff` macro, which are the ones in which we are most interested, is as follows:

```
                    Vacation Example
                Evaluate the Choice Design

                      Final Results

                Design                   1
                Choice Sets             36
                Alternatives             6
                Parameters              35
                Maximum Parameters     180
                D-Efficiency             0
                D-Error                  .
```

```
                          Vacation Example
                     Evaluate the Choice Design

                                                                    Standard
 n Variable Name              Label                       Variance DF   Error

 1 Hawaii                     Hawaii                       1.53333  1  1.23828
 2 Alaska                     Alaska                       1.53545  1  1.23913
 3 Mexico                     Mexico                       1.53545  1  1.23913
 4 California                 California                   1.53333  1  1.23828
 5 Maine                      Maine                        1.53545  1  1.23913
 6 Home                       Home                          .       0   .
 7 Alaska_0                   Alaska,    0                  .       0   .
 8 Alaska_999                 Alaska,  999                 1.20000  1  1.09545
 9 Alaska_1249                Alaska, 1249                 1.20000  1  1.09545
10 Alaska_1499                Alaska, 1499                  .       0   .
11 California_0               California,    0              .       0   .
12 California_999             California,  999             1.20000  1  1.09545
13 California_1249            California, 1249             1.20000  1  1.09545
14 California_1499            California, 1499              .       0   .
15 Hawaii_0                   Hawaii,    0                  .       0   .
16 Hawaii_999                 Hawaii,  999                 1.20000  1  1.09545
17 Hawaii_1249                Hawaii, 1249                 1.20000  1  1.09545
18 Hawaii_1499                Hawaii, 1499                  .       0   .
19 Home_0                     Home,    0                    .       0   .
20 Home_999                   Home,  999                    .       0   .
21 Home_1249                  Home, 1249                    .       0   .
22 Home_1499                  Home, 1499                    .       0   .
23 Maine_0                    Maine,    0                   .       0   .
24 Maine_999                  Maine,  999                  1.20000  1  1.09545
25 Maine_1249                 Maine, 1249                  1.20000  1  1.09545
26 Maine_1499                 Maine, 1499                   .       0   .
27 Mexico_0                   Mexico,    0                  .       0   .
28 Mexico_999                 Mexico,  999                 1.20000  1  1.09545
29 Mexico_1249                Mexico, 1249                 1.20000  1  1.09545
30 Mexico_1499                Mexico, 1499                  .       0   .
31 AlaskaBeach                Alaska, Beach                1.20635  1  1.09834
32 AlaskaHome                 Alaska, Home                  .       0   .
33 AlaskaLake                 Alaska, Lake                 1.20635  1  1.09834
34 AlaskaMountains            Alaska, Mountains             .       0   .
35 CaliforniaBeach            California, Beach            1.20000  1  1.09545
36 CaliforniaHome             California, Home              .       0   .
37 CaliforniaLake             California, Lake             1.20000  1  1.09545
38 CaliforniaMountains        California, Mountains         .       0   .
39 HawaiiBeach                Hawaii, Beach                1.20000  1  1.09545
40 HawaiiHome                 Hawaii, Home                  .       0   .
41 HawaiiLake                 Hawaii, Lake                 1.20000  1  1.09545
42 HawaiiMountains            Hawaii, Mountains             .       0   .
```

```
 43 HomeBeach                 Home, Beach                    .      0   .
 44 HomeHome                  Home, Home                     .      0   .
 45 HomeLake                  Home, Lake                     .      0   .
 46 HomeMountains             Home, Mountains                .      0   .

 47 MaineBeach                Maine, Beach                1.20635   1  1.09834
 48 MaineHome                 Maine, Home                    .      0   .
 49 MaineLake                 Maine, Lake                 1.20635   1  1.09834
 50 MaineMountains            Maine, Mountains               .      0   .

 51 MexicoBeach               Mexico, Beach               1.20635   1  1.09834
 52 MexicoHome                Mexico, Home                   .      0   .
 53 MexicoLake                Mexico, Lake                1.20635   1  1.09834
 54 MexicoMountains           Mexico, Mountains              .      0   .

 55 AlaskaBed___Breakfast     Alaska, Bed & Breakfast     1.20000   1  1.09545
 56 AlaskaCabin               Alaska, Cabin               1.20000   1  1.09545
 57 AlaskaHome2               Alaska, Home                   .      0   .
 58 AlaskaHotel               Alaska, Hotel                  .      0   .

 59 CaliforniaBed___Breakfast California, Bed & Breakfast 1.20000   1  1.09545
 60 CaliforniaCabin           California, Cabin           1.20000   1  1.09545
 61 CaliforniaHome2           California, Home               .      0   .
 62 CaliforniaHotel           California, Hotel              .      0   .

 63 HawaiiBed___Breakfast     Hawaii, Bed & Breakfast     1.20000   1  1.09545
 64 HawaiiCabin               Hawaii, Cabin               1.20000   1  1.09545
 65 HawaiiHome2               Hawaii, Home                   .      0   .
 66 HawaiiHotel               Hawaii, Hotel                  .      0   .

 67 HomeBed___Breakfast       Home, Bed & Breakfast          .      0   .
 68 HomeCabin                 Home, Cabin                    .      0   .
 69 HomeHome2                 Home, Home                     .      0   .
 70 HomeHotel                 Home, Hotel                    .      0   .

 71 MaineBed___Breakfast      Maine, Bed & Breakfast      1.20000   1  1.09545
 72 MaineCabin                Maine, Cabin                1.20000   1  1.09545
 73 MaineHome2                Maine, Home                    .      0   .
 74 MaineHotel                Maine, Hotel                   .      0   .

 75 MexicoBed___Breakfast     Mexico, Bed & Breakfast     1.20000   1  1.09545
 76 MexicoCabin               Mexico, Cabin               1.20000   1  1.09545
 77 MexicoHome2               Mexico, Home                   .      0   .
 78 MexicoHotel               Mexico, Hotel                  .      0   .
                                                                   ==
                                                                   35
```

We see estimable parameters for the five destinations, but not for the stay at home alternative. This is because the last level, the home alternative, is a reference level. For each destination/attribute combination, which are the alternative-specific effects, we see two estimable parameters. We also see two more lines with zero degrees of freedom. One corresponds to the last level of the attribute. This is the usual reference term. It is a linear combination of terms that come before. For example, the reference level for price of Alaska, `Alaska_1499` is equal to `Alaska - Alaska_999 - Alaska_1249`. The other zero degree of freedom term corresponds to staying home. Consider, for example, the price of 0 (the price of staying home) and the Alaska destination. The indicator variable for this term consists of all zeros since the price of 0 never happens with Alaska. So there is no parameter to estimate. You can prove these things to yourself by examining the coded design. The `%ChoicEff` macro makes it available in a SAS data set called `tmp_cand`. This data set is not considered to be one of the "normal" output data sets from the macro, so no note is created about its creation. However, it is always there in case you want to look at it. You can verify that `Alaska_1499` is equal to `Alaska - Alaska_999 - Alaska_1249` as follows:

```
proc reg data=tmp_cand;
   model Alaska_1499 = Alaska Alaska_999 Alaska_1249 / noint;
   run; quit;
```

The results are as follows:

```
                              Vacation Example
                          Evaluate the Choice Design

                             The REG Procedure
                               Model: MODEL1
                  Dependent Variable: Alaska_1499 Alaska, 1499

                    Number of Observations Read         216
                    Number of Observations Used         216

                NOTE: No intercept in model. R-Square is redefined.

                            Analysis of Variance

                                      Sum of           Mean
  Source                   DF        Squares         Square    F Value    Pr > F

  Model                     3       12.00000        4.00000      Infty    <.0001
  Error                   213              0              0
  Uncorrected Total       216       12.00000

               Root MSE                    0    R-Square     1.0000
               Dependent Mean        0.05556    Adj R-Sq     1.0000
               Coeff Var                   0

                            Parameter Estimates

                                     Parameter      Standard
 Variable      Label          DF      Estimate         Error    t Value   Pr > |t|

 Alaska        Alaska          1       1.00000             0      Infty     <.0001
 Alaska_999    Alaska,  999    1      -1.00000             0     -Infty     <.0001
 Alaska_1249   Alaska, 1249    1      -1.00000             0     -Infty     <.0001
```

You can verify that Alaska_0 is all zero as follows:

```
proc freq data=tmp_cand;
   tables Alaska_0;
   run;
```

The results are as follows:

```
                              Vacation Example
                          Evaluate the Choice Design

                             The FREQ Procedure

                               Alaska,    0

                                               Cumulative    Cumulative
             Alaska_0    Frequency     Percent     Frequency      Percent
             ------------------------------------------------------------
                    0         216      100.00           216       100.00
```

You can similarly verify the causes of the 0 *df* for the other terms in the model.

The standard errors for most of the alternative-specific effects are 1.09545, but a few are a bit higher. They correspond to the scenery attributes for Alaska, Maine, and Mexico, which are our nonorthogonal factors. This design looks quite good. Everything that should be estimable in an alternative-specific effects model is estimable, and all of the standard errors are of a similar magnitude.

You can run the %ChoicEff macro one more time and get closer to just a listing of the estimable terms as follows:

```
title2 'Evaluate the Choice Design';
%choiceff(data=sasuser.VacationChDes,/* candidate set of choice sets        */
          init=sasuser.VacationChDes(keep=set), /* select these sets        */
          intiter=0,                 /* evaluate without internal iterations */
                                     /* alternative-specific effects model   */
                                     /* ref level for place is 'Home'        */
                                     /* order=data - do not sort levels      */
          model=class(place / zero='Home' order=data)
                                     /* ref level for place is 'Home'        */
                                     /* ref level for price is 0             */
                                     /* ref level for scene is 'Home'        */
                                     /* ref level for lodge is 'Home'        */
                                     /* order=formatted - sort levels        */
                class(place * price place * scene place * lodge /
                      zero='Home' '0' 'Home' 'Home' order=formatted) /
                lprefix=0            /* lpr=0 labels created from just levels*/
                cprefix=0            /* cpr=0 names created from just levels */
                separators=' ' ', ',/* use comma sep to build interact terms*/
          nsets=36,                  /* number of choice sets                */
          nalts=6,                   /* number of alternatives               */
          beta=zero)                 /* assumed beta vector, Ho: b=0         */
```

The results of this step are not shown, but there are 35 model *df* like before and many fewer zeros. The macro displays a list of the terms with zero *df*. In this case, the list is as follows:

```
Alaska_1499 California_1499 Hawaii_1499 Maine_1499 Mexico_1499 AlaskaMountains
CaliforniaMountains HawaiiMountains MaineMountains MexicoMountains AlaskaHotel
CaliforniaHotel HawaiiHotel MaineHotel MexicoHotel
```

You can incorporate this list into the macro as follows:

```
title2 'Evaluate the Choice Design';

%choiceff(data=sasuser.VacationChDes,/* candidate set of choice sets       */
          init=sasuser.VacationChDes(keep=set), /* select these sets       */
          intiter=0,                /* evaluate without internal iterations */
                                    /* alternative-specific effects model   */
                                    /* ref level for place is 'Home'        */
                                    /* order=data - do not sort levels      */
          model=class(place / zero='Home' order=data)
                                    /* ref level for place is 'Home'        */
                                    /* ref level for price is 0             */
                                    /* ref level for scene is 'Home'        */
                                    /* ref level for lodge is 'Home'        */
                                    /* order=formatted - sort levels        */
                class(place * price place * scene place * lodge /
                      zero='Home' '0' 'Home' 'Home' order=formatted) /
                lprefix=0           /* lpr=0 labels created from just levels*/
                cprefix=0           /* cpr=0 names created from just levels */
                separators=' ' ', ',/* use comma sep to build interact terms*/

          nsets=36,                 /* number of choice sets                */
          nalts=6,                  /* number of alternatives               */
                                    /* extra model terms to drop from model */
          drop=Alaska_1499 California_1499 Hawaii_1499 Maine_1499
               Mexico_1499 AlaskaMountains CaliforniaMountains
               HawaiiMountains MaineMountains MexicoMountains AlaskaHotel
               CaliforniaHotel HawaiiHotel MaineHotel MexicoHotel,
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

Most of the results are as follows:

```
                         Vacation Example
                     Evaluate the Choice Design

              Design   Iteration  D-Efficiency       D-Error
              ----------------------------------------------
                   1       0           1.18690 *     0.84253
```

```
                                   Vacation Example
                              Evaluate the Choice Design

                                     Final Results

                              Design                   1
                              Choice Sets             36
                              Alternatives             6
                              Parameters              35
                              Maximum Parameters     180
                              D-Efficiency        1.1869
                              D-Error             0.8425

                                   Vacation Example
                              Evaluate the Choice Design

                                                                                  Standard
     n Variable Name              Label                        Variance DF    Error

     1 Hawaii                     Hawaii                         1.53333  1  1.23828
     2 Alaska                     Alaska                         1.53545  1  1.23913
     3 Mexico                     Mexico                         1.53545  1  1.23913
     4 California                 California                     1.53333  1  1.23828
     5 Maine                      Maine                          1.53545  1  1.23913
     6 Alaska_999                 Alaska,  999                   1.20000  1  1.09545
     7 Alaska_1249                Alaska, 1249                   1.20000  1  1.09545
     8 California_999             California,  999               1.20000  1  1.09545
     9 California_1249            California, 1249               1.20000  1  1.09545
    10 Hawaii_999                 Hawaii,  999                   1.20000  1  1.09545
    11 Hawaii_1249                Hawaii, 1249                   1.20000  1  1.09545
    12 Maine_999                  Maine,  999                    1.20000  1  1.09545
    13 Maine_1249                 Maine, 1249                    1.20000  1  1.09545
    14 Mexico_999                 Mexico,  999                   1.20000  1  1.09545
    15 Mexico_1249                Mexico, 1249                   1.20000  1  1.09545
    16 AlaskaBeach                Alaska, Beach                  1.20635  1  1.09834
    17 AlaskaLake                 Alaska, Lake                   1.20635  1  1.09834
    18 CaliforniaBeach            California, Beach              1.20000  1  1.09545
    19 CaliforniaLake             California, Lake               1.20000  1  1.09545
    20 HawaiiBeach                Hawaii, Beach                  1.20000  1  1.09545
    21 HawaiiLake                 Hawaii, Lake                   1.20000  1  1.09545
    22 MaineBeach                 Maine, Beach                   1.20635  1  1.09834
    23 MaineLake                  Maine, Lake                    1.20635  1  1.09834
    24 MexicoBeach                Mexico, Beach                  1.20635  1  1.09834
    25 MexicoLake                 Mexico, Lake                   1.20635  1  1.09834
    26 AlaskaBed___Breakfast      Alaska, Bed & Breakfast        1.20000  1  1.09545
    27 AlaskaCabin                Alaska, Cabin                  1.20000  1  1.09545
    28 CaliforniaBed___Breakfast California, Bed & Breakfast    1.20000  1  1.09545
    29 CaliforniaCabin            California, Cabin              1.20000  1  1.09545
```

```
30 HawaiiBed___Breakfast     Hawaii, Bed & Breakfast      1.20000  1  1.09545
31 HawaiiCabin               Hawaii, Cabin                1.20000  1  1.09545
32 MaineBed___Breakfast      Maine, Bed & Breakfast       1.20000  1  1.09545
33 MaineCabin                Maine, Cabin                 1.20000  1  1.09545
34 MexicoBed___Breakfast     Mexico, Bed & Breakfast      1.20000  1  1.09545
35 MexicoCabin               Mexico, Cabin                1.20000  1  1.09545
                                                                  ==
                                                                  35
```

Now we have all of the final 35 parameters. We used the `drop=` option to drop the extra terms. It is often the case with a choice model with a constant alternative that it is hard to write a model that precisely creates all of the right terms with nothing extra. The `drop=` option gives you a way to to remove the extra terms.

These results look good. The variances for Alaska, Mexico, and Maine are slightly larger than the variances for Hawaii and California. Similarly, the variances for the scenery parameters for Alaska, Mexico, and Maine are larger than the variances for the other attributes. This is because we have a small amount of nonorthogonality in those attributes. You can see that this effect is slight.

There is one more test that should be run before a design is used. The following `%MktDups` macro step checks the design to see if any choice sets are duplicates of any other choice sets:

```
%mktdups(branded, data=sasuser.VacationChDes,
         nalts=6, factors=place price scene lodge)
```

The results are as follows:

```
Design:          Branded
Factors:         place price scene lodge
                 Place
                 Lodge Price Scene
Duplicate Sets:  0
```

The first line of the table tells us that this is a branded design as opposed to a generic design (bundles of attributes with no brands). The second line tells us the factors as specified in the `factors=` option. These are followed by the actual variable names for the factors. The last line reports the number of duplicates. In this case, there are no duplicate choice sets. If there are duplicate choice sets, then changing the random number seed might help. Changing other aspects of the design or the approach for making the design might also help.

Generating the Questionnaire

The following DATA step produces the questionnaires:

```
title;
proc sort data=sasuser.VacationLinDesBlckd; by block; run;

options ls=80 ps=60 nodate nonumber;

data _null_;
   array dests[&mm1] $ 10 _temporary_ ('Hawaii' 'Alaska' 'Mexico'
                                        'California' 'Maine');
   array prices[3]   $  5 _temporary_ ('$999' '$1249' '$1499');
   array scenes[3]   $ 13 _temporary_
                     ('the Mountains' 'a Lake' 'the Beach');
   array lodging[3]  $ 15 _temporary_
                     ('Cabin' 'Bed & Breakfast' 'Hotel');
   array x[15];
   file print linesleft=ll;

   set sasuser.VacationLinDesBlckd;
   by block;

   if first.block then do;
      choice = 0;
      put _page_;
      put @50 'Form: ' block  ' Subject: ________' //;
      end;
   choice + 1;

   if ll < 19 then put _page_;
   put choice 2. ') Circle your choice of '
       'vacation destinations:' /;
   do dest = 1 to &mm1;
      put '    ' dest 1. ') ' dests[dest]
          +(-1) ', staying in a ' lodging[x[dest]]
          'near ' scenes[x[&mm1 + dest]] +(-1) ',' /
          '       with a package cost of '
         prices[x[2 * &mm1 + dest]] +(-1) '.' /;
      end;
   put "    &m) Stay at home this year." /;
   run;
```

They are then copied and the data are collected. In this design, there are five destinations, and each destination has three attributes. Each destination name is accessed from the array `dests`. Note that destination is not a factor in the design; it is a bin into which the attributes are grouped. The factors in the design are named in the statement `array x[15]`, which is a short-hand notation for `array x[15] x1-x15`. The first five factors are used for the lodging attribute of the five destinations. The actual descriptions of lodging are accessed by `lodging[x[dest]]`. The variable `Dest` varies from 1 to 5 destinations, so `x[dest]` extracts the levels for the `Dest` destination. Similarly for scenery, `scenes[x[&mm1 + dest]]` extracts the descriptions of the scenery. The index `&mm1 + dest` accesses factors 6 through 10, and `x[&mm1 + dest]` indexes the `scenes` array. For prices, `prices[x[2 * &mm1 + dest]]`, the index `2 * &mm1 + dest` accesses the factors 11 through 15.

The first two choice sets are as follows:

```
                              Vacation Example
                                           Form: 1  Subject: ________


   1) Circle your choice of vacation destinations:

      1) Hawaii, staying in a Cabin near the Beach,
         with a package cost of $999.

      2) Alaska, staying in a Hotel near the Mountains,
         with a package cost of $999.

      3) Mexico, staying in a Hotel near the Beach,
         with a package cost of $1249.

      4) California, staying in a Cabin near a Lake,
         with a package cost of $1249.

      5) Maine, staying in a Cabin near the Mountains,
         with a package cost of $1249.

      6) Stay at home this year.

   2) Circle your choice of vacation destinations:

      1) Hawaii, staying in a Cabin near the Beach,
         with a package cost of $999.

      2) Alaska, staying in a Bed & Breakfast near a Lake,
         with a package cost of $1499.

      3) Mexico, staying in a Cabin near a Lake,
         with a package cost of $999.

      4) California, staying in a Bed & Breakfast near the Mountains,
         with a package cost of $999.

      5) Maine, staying in a Hotel near the Beach,
         with a package cost of $1249.

      6) Stay at home this year.
```

In practice, data collection is typically much more elaborate than this. It might involve art work or photographs, and the choice sets might be presented and the data might be collected through personal interview or over the Web. However the choice sets are presented and the data are collected, the essential elements remain the same. Subjects are shown a set of alternatives and are asked to make a choice, then they go on to the next set.

Entering and Processing the Data

The data are read as follows:

```
title 'Vacation Example';

data results;
   input Subj Form (choose1-choose&n) (1.) @@;
   datalines;
  1  1 111353313351554151   2  2 344113155513111413   3  1 132353331151534151
  4  2 341133131523331143   5  1 142153111151334143   6  2 344114111543131151
  7  1 141343111311154154   8  2 344113111343121111   9  1 141124131151342155
 10  2 344113131523131141  11  1 311423131353524144  12  2 332123151413331151
 13  1 311244331352134155  14  2 341114111543131153  15  1 141253111351344151
 16  2 344135131323331143  17  1 142123313154132141  18  2 542113151323131141
 19  1 145314111311144111  20  2 344111131313431143  21  1 133343131313432145
.
.
.
;
```

Data from a total of 200 subjects are collected, 100 per form.

Next, we use the `%MktMerge` macro as follows to combine the data and design and create the variable `c`, indicating whether each alternative is a first choice or a subsequent choice:

```
%mktmerge(design=sasuser.VacationChDes, data=results, out=res2, blocks=form,
          nsets=&n, nalts=&m, setvars=choose1-choose&n)

proc print data=res2(obs=12);
   id subj form set; by subj form set;
   run;
```

This macro takes the `design=sasuser.VacationChDes` experimental design, merges it with the `data=result` data set, creating the `out=res2` output data set. The `Results` data set contains the variable `Form` that contains the block number. Since there are two blocks, this variable must have values of 1 and 2. This variable must be specified in the `blocks=` option. The experiment has `nsets=&n` choice sets, `nalts=6` alternatives, and the variables `setvars=choose1-choose&n` contain the numbers of the chosen alternatives. The output data set `Res2` has 21,600 observations (200 subjects who each saw 18 choice sets with 6 alternatives).

The first two choice sets are as follows:

```
                              Vacation Example

 Subj    Form    Set    Place         Lodge              Scene         Price    c

   1       1      1     Hawaii        Cabin              Beach           999    1
                        Alaska        Hotel              Mountains       999    2
                        Mexico        Hotel              Beach          1249    2
                        California    Cabin              Lake           1249    2
                        Maine         Cabin              Mountains      1249    2
                        Home          Home               Home              0    2

   1       1      2     Hawaii        Cabin              Beach           999    1
                        Alaska        Bed & Breakfast    Lake           1499    2
                        Mexico        Cabin              Lake            999    2
                        California    Bed & Breakfast    Mountains       999    2
                        Maine         Hotel              Beach          1249    2
                        Home          Home               Home              0    2
```

Binary Coding

One more thing must be done to these data before they can be analyzed. The binary design matrix is coded for each effect. This can be done with PROC TRANSREG as follows:

```
proc transreg design=5000 data=res2 nozeroconstant norestoremissing;
   model class(place / zero=none order=data)
         class(price scene lodge / zero=none order=formatted) /
         lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   id subj set form c;
   run;
```

The `design` option specifies that no model is fit; the procedure is just being used to code a design. When `design` is specified, dependent variables are not required. Optionally, `design` can be followed by "=*n*" where *n* is the number of observations to process at one time. By default, PROC TRANSREG codes all observations in one big group. For very large data sets, this can consume large amounts of memory and time. Processing blocks of smaller numbers of observations is more computationally efficient. The option `design=5000` processes observations in blocks of 5000. For smaller computers, try something like `design=1000`.

The `nozeroconstant` and `norestoremissing` options are not necessary for this example but are included here because sometimes they are very helpful in coding choice models. The `nozeroconstant` option specifies that if the coding creates a constant variable, it should not be zeroed. The `nozeroconstant` option should always be specified when you specify `design=`*n* because the last group of observations might be small and might contain constant variables. The `nozeroconstant` option is also important if you do something like coding `by subj set` because sometimes an attribute is constant within a choice set. The norestoremissing option specifies that missing values should not be restored when the out= data set is created. By default, the coded class variable contains a row of missing values for observations in which the class variable is missing. With the norestoremissing option, these observations contain a row of zeros instead. This option is useful when there is a constant alternative indicated by missing values. Both of these options, like almost all options in PROC TRANSREG, can be abbreviated to three characters (noz and nor).

The model statement names the variables to code and provides information about how they should be coded. The specification class(place / ...) specifies that the variable Place is a classification variable and requests a binary coding. The zero=none option creates binary variables for all categories. The order=data option sorts the levels into the order they are encountered in the data set. It is specified so 'Home' is the last destination in the analysis, as it is in the data set. The class(price scene lodge / ...) specification names the variables Price, Scene, and Lodge as categorical variables and creates binary variables for all of the levels of all of the variables. The levels are sorted into order based on their formatted values. The lprefix=0 option specifies that when labels are created for the binary variables, zero characters of the original variable name should be used as a prefix. This means that the labels are created only from the level values. For example, 'Mountains' and 'Bed & Breakfast' are created as labels not 'Scene Mountains' and 'Lodge Bed & Breakfast'.

An output statement names the output data set and drops variables that are not needed. These variables do not have to be dropped. However, since they are variable names that are often found in special data set types, PROC PHREG displays warnings when it finds them. Dropping the variables prevents the warnings. Finally, the id statement names the additional variables that we want copied from the input to the output data set. The following steps display the first coded choice set:

```
proc print data=coded(obs=6);
   id place;
   var subj set form c price scene lodge;
   run;

proc print data=coded(obs=6) label;
   var pl:;
   run;

proc print data=coded(obs=6) label;
   id place;
   var sc:;
   run;

proc print data=coded(obs=6) label;
   id place;
   var lo: pr:;
   run;
```

The results are as follows:

```
                              Vacation Example

      Place          Subj     Set     Form     c     Price     Scene         Lodge

      Hawaii            1       1        1     1       999     Beach         Cabin
      Alaska            1       1        1     2       999     Mountains     Hotel
      Mexico            1       1        1     2      1249     Beach         Hotel
      California        1       1        1     2      1249     Lake          Cabin
      Maine             1       1        1     2      1249     Mountains     Cabin
      Home              1       1        1     2         0     Home          Home
                              Vacation Example

   Obs    Hawaii    Alaska    Mexico    California    Maine    Home    Place

     1       1         0         0           0          0        0     Hawaii
     2       0         1         0           0          0        0     Alaska
     3       0         0         1           0          0        0     Mexico
     4       0         0         0           1          0        0     California
     5       0         0         0           0          1        0     Maine
     6       0         0         0           0          0        1     Home
                              Vacation Example

          Place          Beach    Home    Lake    Mountains    Scene

          Hawaii           1        0       0         0        Beach
          Alaska           0        0       0         1        Mountains
          Mexico           1        0       0         0        Beach
          California       0        0       1         0        Lake
          Maine            0        0       0         1        Mountains
          Home             0        1       0         0        Home
                              Vacation Example

                    Bed &
 Place            Breakfast    Cabin    Home    Hotel    Lodge      0    999    1249    1499    Price

 Hawaii               0          1        0       0      Cabin      0     1       0       0      999
 Alaska               0          0        0       1      Hotel      0     1       0       0      999
 Mexico               0          0        0       1      Hotel      0     0       1       0     1249
 California           0          1        0       0      Cabin      0     0       1       0     1249
 Maine                0          1        0       0      Cabin      0     0       1       0     1249
 Home                 0          0        1       0      Home       1     0       0       0        0
```

The coded design consists of binary variables for destinations Hawaii — Home, scenery Beach — Mountains, lodging Bed & Breakfast — Hotel, and price 0 — 1499. For example, in the last panel of the first choice set, the Cabin column has a 1 for Hawaii since Hawaii has Cabin lodging in this choice set. The Cabin column has a 0 for Alaska since Alaska does not have Cabin lodging in this choice set. These binary variables form the independent variables in the analysis.

Note that we are fitting a model with *generic attributes*. Generic attributes are assumed to be the same for all alternatives. For example, our model is structured so that the part-worth utility for being on a lake is the same for Hawaii, Alaska, and all of the other destinations. Similarly, the part-worth utilities for the different prices do not depend on the destinations. In contrast, on page 386, using the same data, we code alternative-specific effects where the part-worth utilities are allowed by the model to be different for each of the destinations.

PROC PHREG is run to fit the choice model as follows:

```
proc phreg data=coded brief;
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;
```

We specify the `&_trgind` macro variable for the `model` statement independent variable list. PROC TRANSREG automatically creates this macro variable. It contains the list of coded independent variables generated by the procedure. This is so you do not have to figure out what names TRANSREG created and specify them. In this case, PROC TRANSREG sets `&_trgind` to contain the following list:

```
PlaceHawaii PlaceAlaska PlaceMexico PlaceCalifornia PlaceMaine PlaceHome
Price0 Price999 Price1249 Price1499 SceneBeach SceneHome SceneLake
SceneMountains LodgeBed___Breakfast LodgeCabin LodgeHome LodgeHotel
```

The analysis is stratified by subject and choice set. Each stratum consists of a set of alternatives from which a subject made one choice. In this example, each stratum consists of six alternatives, one of which is chosen and five of which are not chosen. (Recall that we specified `%phchoice(on)` on page 287 to customize the output from PROC PHREG.) The full table of the strata is quite large with one line for each of the 3600 strata, so the `brief` option is specified in the PROC PHREG statement. This option produces a brief summary of the strata. In this case, we see there are 3600 choice sets that all fit one response pattern. Each consisted of 6 alternatives, 1 of which is chosen and 5 of which are not chosen. There should be one pattern for all choice sets in an example like this one—the number of alternatives, number of chosen alternatives, and the number not chosen should be constant.

The results are as follows:

```
                          Vacation Example

                        The PHREG Procedure

                         Model Information

                  Data Set                 WORK.CODED
                  Dependent Variable       c
                  Censoring Variable       c
                  Censoring Value(s)       2
                  Ties Handling            BRESLOW

                Number of Observations Read       21600
                Number of Observations Used       21600

      Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                 Number of       Number of          Chosen          Not
      Pattern      Choices    Alternatives    Alternatives       Chosen

            1         3600               6               1            5

                         Convergence Status

             Convergence criterion (GCONV=1E-8) satisfied.

                       Model Fit Statistics

                                 Without           With
                Criterion     Covariates     Covariates

                -2 LOG L       12900.668       6257.752
                AIC            12900.668       6279.752
                SBC            12900.668       6347.827

                Testing Global Null Hypothesis: BETA=0

        Test                 Chi-Square       DF     Pr > ChiSq

        Likelihood Ratio      6642.9164       11         <.0001
        Score                 5858.3798       11         <.0001
        Wald                  2482.5118       11         <.0001
```

```
                          Multinomial Logit Parameter Estimates

                                Parameter      Standard
                          DF     Estimate         Error    Chi-Square    Pr > ChiSq

    Hawaii                 1      3.50429       0.45819       58.4934        <.0001
    Alaska                 1      0.62029       0.46624        1.7699        0.1834
    Mexico                 1      2.81487       0.45955       37.5193        <.0001
    California             1      2.13549       0.46027       21.5263        <.0001
    Maine                  1      1.53470       0.46220       11.0253        0.0009
    Home                   0            0             .             .             .
      0                    0            0             .             .             .
     999                   1      3.56656       0.08849     1624.2978        <.0001
    1249                   1      1.40145       0.08293      285.6189        <.0001
    1499                   0            0             .             .             .
    Beach                  1      1.34191       0.06410      438.2880        <.0001
    Home                   0            0             .             .             .
    Lake                   1      0.67993       0.06981       94.8542        <.0001
    Mountains              0            0             .             .             .
    Bed & Breakfast        1      0.64972       0.05363      146.7874        <.0001
    Cabin                  1     -1.41463       0.07581      348.1654        <.0001
    Home                   0            0             .             .             .
    Hotel                  0            0             .             .             .
```

The destinations, from most preferred to least preferred, are Hawaii, Mexico, California, Maine, Alaska, and then stay at home. The utility for lower price is greater than the utility for higher price. The beach is preferred over a lake, which is preferred over the mountains. A bed & breakfast is preferred over a hotel, which is preferred over a cabin. Notice that the coefficients for the constant alternative (home and zero price) are all zero. Also notice that for each factor, destination, price, scenery and accommodations, the coefficient for the last level is always zero. This always occurs when we code with `zero=none`. The last level of each factor is a reference level, and the other coefficients have values relative to this zero. For example, all of the coefficients for the destination are positive relative to the zero for staying at home. For scenery, all of the coefficients are positive relative to the zero for the mountains. For accommodations, the coefficient for cabin is less than the zero for hotel, which is less than the coefficient for bed & breakfast. In some sense, each `class` variable in a choice model with a constant alternative has levels that always have a zero coefficient: the level corresponding to the constant alternative and the level corresponding to the last level. At first, it is reassuring to run the model with all levels represented to see that all the right levels get zeroed. Later, we will eliminate these levels from the output.

Quantitative Price Effect

These data can also be analyzed in a different way. The `Price` variable can be specified directly as a quantitative variable, instead of with indicator variables for a qualitative price effect. You could display the independent variable list and copy and edit it, removing the `Price` indicator variables and adding `Price`.

The following statement displays the list:

```
%put &_trgind;
```

Alternatively, you could run PROC TRANSREG again with the new coding. We use this latter approach, because it is easier, and it lets us illustrate other options. In the previous analysis, there are a number of structural zeros in the parameter estimate results due to the usage of the `zero=none` option in the PROC TRANSREG coding. This is a good thing, particularly for a first attempt at the analysis. It is good to specify `zero=none` and check the results and make sure you have the right pattern of zeros and nonzeros. Later, you can run again excluding some of the structural zeros. This time, we explicitly specify the `'Home'` level in the `zero=` option as the reference level so it is omitted from the `&_trgind` variable list. The first `class` specification specifies `zero='Home'` since there is one variable. The second `class` specification specifies `zero='Home' 'Home'` specifying the reference level for each of the two variables. The variable `Price` is designated as an `identity` variable. The `identity` transformation is the no-transformation option, which is used for variables that need to enter the model with no further manipulations. The `identity` variables are simply copied into the output data set and added to the `&_trgind` variable list. The statement `label price = 'Price'` is specified to explicitly set a label for the `identity` variable price. This is because we explicitly modified PROC PHREG output using `%phchoice(on)` so that the rows of the parameter estimate table are labeled only with variable labels not variable names. A label for `Price` must be explicitly specified in order for the output to contain a label for the price effect. The following steps perform the coding and the analysis:

```
proc transreg design data=res2 nozeroconstant norestoremissing;
   model class(place / zero='Home' order=data) identity(price)
         class(scene lodge / zero='Home' 'Home' order=formatted) /
         lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   label price = 'Price';
   id subj set form c;
   run;

proc phreg data=coded brief;
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;
```

The results are as follows:

```
                              Vacation Example

                            The PHREG Procedure

                             Model Information

                   Data Set                   WORK.CODED
                   Dependent Variable         c
                   Censoring Variable         c
                   Censoring Value(s)         2
                   Ties Handling              BRESLOW
```

```
                    Number of Observations Read       21600
                    Number of Observations Used       21600

          Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                     Number of       Number of          Chosen         Not
         Pattern       Choices    Alternatives    Alternatives      Chosen

               1          3600               6               1           5

                               Convergence Status

                 Convergence criterion (GCONV=1E-8) satisfied.

                             Model Fit Statistics

                                   Without           With
                    Criterion     Covariates     Covariates

                    -2 LOG L       12900.668       6295.152
                    AIC            12900.668       6315.152
                    SBC            12900.668       6377.039

                    Testing Global Null Hypothesis: BETA=0

            Test                 Chi-Square       DF     Pr > ChiSq

            Likelihood Ratio      6605.5164       10         <.0001
            Score                 5750.9220       10         <.0001
            Wald                  2483.9241       10         <.0001

                     Multinomial Logit Parameter Estimates

                                Parameter      Standard
                        DF       Estimate         Error    Chi-Square    Pr > ChiSq

    Hawaii               1       14.27118       0.50198      808.2623        <.0001
    Alaska               1       11.44532       0.49063      544.1855        <.0001
    Mexico               1       13.56216       0.49955      737.0457        <.0001
    California           1       12.94025       0.49430      685.3359        <.0001
    Maine                1       12.36405       0.49553      622.5618        <.0001
    Price                1       -0.00740     0.0001770     1747.2333        <.0001
    Beach                1        1.33978       0.06458      430.4561        <.0001
    Lake                 1        0.71161       0.07131       99.5777        <.0001
    Mountains            0              0             .             .             .
    Bed & Breakfast      1        0.66233       0.05319      155.0604        <.0001
    Cabin                1       -1.33467       0.07353      329.4356        <.0001
    Hotel                0              0             .             .             .
```

The results of the two different analyses are similar. The coefficients for the destinations all increase by a nonconstant amount (approximately 10.8) but the pattern is the same. There is still a negative effect for price. Also, the fit of this model is slightly worse, Chi-Square = 6605.5164, compared to the previous value of 6642.9164 (bigger values mean better fit), because price has one fewer parameter.

Quadratic Price Effect

Previously, we saw price treated as a qualitative factor with two parameters and two *df*, then we saw price treated as a quantitative factor with one parameter and one *df*. Alternatively, we could treat price as quantitative and add a *quadratic* price effect (price squared). Like treating price as a qualitative factor, there are two parameters and two *df* for price. First, we create `PriceL`, the linear price term by centering the original price and dividing by the price increment (250). This maps (999, 1249, 1499) to (−1, 0, 1). Next, we run PROC TRANSREG and PROC PHREG with the new price variables as follows:

```
data res3;
   set res2;
   PriceL = price;
   if price then pricel = (price - 1249) / 250;
   run;

proc transreg design=5000 data=res3 nozeroconstant norestoremissing;
   model class(place / zero='Home' order=data)
         pspline(pricel / degree=2)
         class(scene lodge / zero='Home' 'Home' order=formatted) /
         lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   label pricel = 'Price';
   id subj set form c;
   run;
```

The `pspline` or polynomial spline expansion with the `degree=2` option replaces the variable `PriceL` with two coded variables, `PriceL_1` (which is the same as the original `PriceL`) and `PriceL_2` (which is `PriceL` squared). A `degree=2` spline with no knots (neither `knots=` nor `nknots=` is specified) simply expands the variable into a quadratic polynomial.

The following step fits the model:

```
proc phreg data=coded brief;
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;
```

This step produced the following results:

```
                              Vacation Example

                            The PHREG Procedure

                             Model Information

                      Data Set                 WORK.CODED
                      Dependent Variable       c
                      Censoring Variable       c
                      Censoring Value(s)       2
                      Ties Handling            BRESLOW

                    Number of Observations Read       21600
                    Number of Observations Used       21600

        Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                 Number of        Number of          Chosen          Not
        Pattern      Choices    Alternatives    Alternatives      Chosen

              1         3600               6               1           5

                             Convergence Status

                 Convergence criterion (GCONV=1E-8) satisfied.

                           Model Fit Statistics

                                  Without           With
                    Criterion     Covariates     Covariates

                    -2 LOG L       12900.668       6257.752
                    AIC            12900.668       6279.752
                    SBC            12900.668       6347.827

                    Testing Global Null Hypothesis: BETA=0

            Test                 Chi-Square       DF     Pr > ChiSq

            Likelihood Ratio      6642.9164       11         <.0001
            Score                 5858.3798       11         <.0001
            Wald                  2482.5118       11         <.0001
```

```
                        Multinomial Logit Parameter Estimates

                             Parameter      Standard
                        DF    Estimate         Error    Chi-Square    Pr > ChiSq

    Hawaii               1     4.90574       0.45379      116.8713        <.0001
    Alaska               1     2.02174       0.46010       19.3081        <.0001
    Mexico               1     4.21633       0.45427       86.1476        <.0001
    California           1     3.53695       0.45507       60.4085        <.0001
    Maine                1     2.93615       0.45761       41.1683        <.0001
    Price 1              1    -1.78328       0.04425     1624.2978        <.0001
    Price 2              1     0.38183       0.06263       37.1732        <.0001
    Beach                1     1.34191       0.06410      438.2880        <.0001
    Lake                 1     0.67993       0.06981       94.8542        <.0001
    Mountains            0           0             .             .             .
    Bed & Breakfast      1     0.64972       0.05363      146.7874        <.0001
    Cabin                1    -1.41463       0.07581      348.1654        <.0001
    Hotel                0           0             .             .             .
```

The fit is exactly the same as when price is treated as qualitative, Chi-Square = 6642.9164. This is because both models are the same except for the different but equivalent 2 *df* codings of price. The coefficients for the destinations in the two models differ by a constant 1.40145. The coefficients for the factors after price are unchanged. The part-worth utility for \$999 is $-1.78328 \times (999 - 1249)/250 + 0.38183 \times ((999 - 1249)/250)^2 = 2.16511$, the part-worth utility for \$1249 is $-1.78328 \times (1249 - 1249)/250 + 0.38183 \times ((1249 - 1249)/250)^2 = 0$, and the part-worth utility for \$1499 is $-1.78328 \times (1499-1249)/250 + 0.38183 \times ((1499-1249)/250)^2 = -1.40145$, which differ from the coefficients when price is treated as qualitative, by a constant -1.40145.

Effects Coding

In the previous analyses, *binary* (1, 0) codings are used for the variables. The next analysis illustrates *effects* (1, 0, −1) coding. The two codings differ in how the final reference level is coded. In binary coding, the reference level is coded with zeros. In effects coding, the reference level is coded with minus ones.

	Binary Coding		Effects Coding	
Levels	One	Two	One	Two
1	1	0	1	0
2	0	1	0	1
3	0	0	-1	-1

In this example, we use a binary coding for the destinations and effects codings for the attributes.

PROC TRANSREG can be used for effects coding. The `effects` option used inside the parentheses after `class` asks for a (0, 1, −1) coding. The `zero=` option specifies the levels that receive the −1's. PROC PHREG is run with almost the same variable list as before, except now the variables for the reference levels, those whose parameters are structural zeros are omitted. Refer back to the parameter estimates table on page 376, a few select lines of which are reproduced next:

```
                          (Some Lines in the)
                  Multinomial Logit Parameter Estimates

                          Parameter      Standard
                   DF      Estimate         Error    Chi-Square    Pr > ChiSq

Home                0             0             .             .             .
   0                0             0             .             .             .
1499                0             0             .             .             .
Home                0             0             .             .             .
Mountains           0             0             .             .             .
Home                0             0             .             .             .
Hotel               0             0             .             .             .
```

Notice that the coefficients for the constant alternative (home and zero price) are all zero. Also notice that for each factor, destination, price, scenery and accommodations, the coefficient for the last level is always zero. In some of the preceding examples, we eliminated the `'Home'` levels by specifying `zero=Home`. Next we will see how to eliminate all of the structural zeros from the parameter estimate table.

First, for each classification variable, we change the level for the constant alternative to missing. (Recall that they were originally missing and we only made them nonmissing to deliberately produce the zero coefficients.) This causes PROC TRANSREG to ignore those levels when constructing indicator variables. When you use this strategy, you must specify the `norestoremissing` option in the PROC TRANSREG statement. During the first stage of design matrix creation, PROC TRANSREG puts zeros in the indicator variables for observations with missing `class` levels. At the end, it replaces the zeros with missings, "restoring the missing values." When the `norestoremissing` option is specified, missing values are not restored and we get zeros in the indicator variables for missing `class` levels, which is usually what we want. The DATA step `if` statements recode the constant levels to missing. Next, in PROC TRANSREG, the reference levels `'Mountains'` and `'Hotel'` are listed in the `zero=` option in the `class` specification as follows:

```
data res4;
   set res3;
   if scene = 0 then scene = .;
   if lodge = 0 then lodge = .;
   run;
```

```
proc transreg design=5000 data=res4 nozeroconstant norestoremissing;
   model class(place / zero='Home' order=data)
         pspline(pricel / degree=2)
         class(scene lodge /
               effects zero='Mountains' 'Hotel' order=formatted) /
         lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   label pricel = 'Price';
   id subj set form c;
   run;
```

Next, the coded data and design matrix are displayed for the first choice set. The coded design matrix begins with five binary columns for the destinations, `'Hawaii'` through `'Maine'`. There is not a column for the stay-at-home destination and the row for stay at home has all zeros in the coded variables. Next is the linear price effect, `'Price 1'`, consisting of 0, 1, and −1. It is followed by the quadratic price effect, `'Price 2'`, which is `'Price 1'` squared. Next are the scenery terms, effects coded. `'Beach'` and `'Lake'` have values of 0 and 1; −1's in the fourth row for the reference level, `'Mountains'`; and zeros in the last row for the stay-at-home alternative. Next are the lodging terms, effects coded. `'Bed & Breakfast'` and `'Cabin'` have values of 0 and 1; −1's in the first, third and fourth row for the reference level, `'Hotel'`; and zeros in the last row for the stay-at-home alternative. The following step displays the lodging terms:

```
proc print data=coded(obs=6) label; run;
```

The results are as follows:

```
                                Vacation Example

                                                 Price Price              Bed &
 Obs Hawaii Alaska Mexico California Maine   1     2   Beach Lake Breakfast Cabin

   1    1      0      0        0        0     -1     1     1     0       0       1
   2    0      1      0        0        0     -1     1    -1    -1      -1      -1
   3    0      0      1        0        0      0     0     1     0      -1      -1
   4    0      0      0        1        0      0     0     0     1       0       1
   5    0      0      0        0        1      0     0    -1    -1       0       1
   6    0      0      0        0        0      0     0     0     0       0       0

    Obs Place        Price    Scene        Lodge        Subj  Set  Form  c

      1 Hawaii        -1   Beach      Cabin              1     1     1   1
      2 Alaska        -1   Mountains  Hotel              1     1     1   2
      3 Mexico         0   Beach      Hotel              1     1     1   2
      4 California     0   Lake       Cabin              1     1     1   2
      5 Maine          0   Mountains  Cabin              1     1     1   2
      6 Home           0           .                 .   1     1     1   2
```

The following step fits the choice model:

```
proc phreg data=coded brief;
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;
```

The results are as follows:

```
                              Vacation Example

                            The PHREG Procedure

                             Model Information

                   Data Set                 WORK.CODED
                   Dependent Variable       c
                   Censoring Variable       c
                   Censoring Value(s)       2
                   Ties Handling            BRESLOW

                 Number of Observations Read        21600
                 Number of Observations Used        21600

       Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                 Number of       Number of          Chosen          Not
       Pattern     Choices    Alternatives    Alternatives       Chosen

             1        3600               6               1            5

                             Convergence Status

                 Convergence criterion (GCONV=1E-8) satisfied.

                           Model Fit Statistics

                                 Without           With
                 Criterion     Covariates     Covariates

                 -2 LOG L       12900.668       6257.752
                 AIC            12900.668       6279.752
                 SBC            12900.668       6347.827

                 Testing Global Null Hypothesis: BETA=0

           Test                 Chi-Square       DF     Pr > ChiSq

           Likelihood Ratio      6642.9164       11         <.0001
           Score                 5858.3798       11         <.0001
           Wald                  2482.5118       11         <.0001
```

```
                     Multinomial Logit Parameter Estimates

                           Parameter      Standard
                    DF      Estimate         Error    Chi-Square    Pr > ChiSq

Hawaii               1       5.32472       0.44985      140.1076        <.0001
Alaska               1       2.44072       0.45690       28.5360        <.0001
Mexico               1       4.63531       0.45052      105.8613        <.0001
California           1       3.95593       0.45176       76.6803        <.0001
Maine                1       3.35513       0.45381       54.6610        <.0001
Price 1              1      -1.78328       0.04425     1624.2978        <.0001
Price 2              1       0.38183       0.06263       37.1732        <.0001
Beach                1       0.66796       0.03582      347.7320        <.0001
Lake                 1       0.00599       0.03922        0.0233        0.8787
Bed & Breakfast      1       0.90469       0.03471      679.2342        <.0001
Cabin                1      -1.15966       0.04650      621.9367        <.0001
```

It is instructive to compare the results of this analysis to the previous analysis on page 381. First, the model fit and chi-square statistics are the same indicating the models are equivalent. The coefficients for the destinations differ by a constant -0.41898, the price coefficients are the same, the scenery coefficients differ by a constant 0.67395, and the lodging coefficients differ by a constant -0.25497. Notice that $-0.41898 + 0 + 0.67395 + -0.25497 = 0$, so the utility for each alternative is unchanged by the different but equivalent codings.

Alternative-Specific Effects

In all of the analyses presented so far in this example, we have assumed that the effects for price, scenery, and accommodations are generic or constant across the different destinations. Equivalently, we assumed that destination does not interact with the attributes. Next, we show a model with *alternative-specific effects* that does not make this assumption. The alternative-specific model allows for different price, scenery and lodging effects for each destination. The coding can be done with PROC TRANSREG using its syntax for interactions. Before we do the coding, let's go back to the design preparation stage and redo it in a more normal fashion so reference levels are omitted from the analysis.

We start by creating the data set `Key` as follows:

```
data key;
   input Place $ 1-10 (Lodge Scene Price) ($);
   datalines;
Hawaii      x1  x6   x11
Alaska      x2  x7   x12
Mexico      x3  x8   x13
California  x4  x9   x14
Maine       x5  x10  x15
.           .   .    .
;
```

This step differs from the one we saw on page 356 only in that now we have a missing value for `Place` for the constant alternative.Next, we use the `%MktRoll` macro to process the design and the `%MktMerge` macro to merge the design and data as follows:

```
%mktroll(design=sasuser.VacationLinDesBlckd, key=key, alt=place,
         out=sasuser.VacationChDes)

%mktmerge(design=sasuser.VacationChDes, data=results, out=res2, blocks=form,
          nsets=&n, nalts=&m, setvars=choose1-choose&n,
          stmts=%str(price = input(put(price, price.), 5.);
                     format scene scene. lodge lodge.;))

proc print data=res2(obs=12); run;
```

The usage of the `%MktRoll` macro is exactly the same as we saw on page 356. The `%MktMerge` macro usage differs from page 371 in that instead of assigning labels and recoding price in a separate DATA step, we now do it directly in the macro. The `stmts=` option is used to add a `price =` assignment statement and `format` statement to the DATA step that merges the two data sets. The statements are included in a `%str( )` macro since they contain semicolons. The first two choice sets are as follows:

```
                                Vacation Example

     Obs   Subj   Form   Set   Place        Lodge             Scene      Price   c

       1     1      1     1    Hawaii       Cabin             Beach        999   1
       2     1      1     1    Alaska       Hotel             Mountains    999   2
       3     1      1     1    Mexico       Hotel             Beach       1249   2
       4     1      1     1    California   Cabin             Lake        1249   2
       5     1      1     1    Maine        Cabin             Mountains   1249   2
       6     1      1     1                                 .         .      .   2
       7     1      1     2    Hawaii       Cabin             Beach        999   1
       8     1      1     2    Alaska       Bed & Breakfast   Lake        1499   2
       9     1      1     2    Mexico       Cabin             Lake         999   2
      10     1      1     2    California   Bed & Breakfast   Mountains    999   2
      11     1      1     2    Maine        Hotel             Beach       1249   2
      12     1      1     2                                 .         .      .   2
```

Notice that the attributes for the constant alternative are all missing. Next, we code with PROC TRANSREG. Since we are using missing values for the constant alternative, we must specify the `norestoremissing` option in the PROC TRANSREG statement. With the `norestoremissing` option, the indicator variables created for missing `class` variable values contain all zeros instead of all missings. First, we specify the variable `Place` as a `class` variable.

Next, we interact `Place` with all of the attributes, `Price`, `Scene`, and `Lodge`, to create the alternative-specific effects as follows:

```
proc transreg design=5000 data=res2 nozeroconstant norestoremissing;
   model class(place / zero=none order=data)
         class(place * price place * scene place * lodge /
               zero=none order=formatted) / lprefix=0 sep=' ' ', ';
   output out=coded(drop=_type_ _name_ intercept);
   id subj set form c;
   run;

proc print data=coded(obs=6) label noobs; run;
```

The coded design matrix consists of:

- five binary columns, `'Hawaii'` through `'Maine'`, for the five destinations,
- fifteen binary columns (5 destinations times 3 prices), `'Alaska, 999'` through `'Mexico, 1499'`, for the alternative-specific price effects,
- fifteen binary columns (5 destinations times 3 sceneries), `'Alaska, Beach'` through `'Mexico, Mountains'`, for the alternative-specific scenery effects,
- fifteen binary columns (5 destinations times 3 lodgings), `'Alaska, Bed & Breakfast'` through `'Mexico, Hotel'`, for the alternative-specific lodging effects.

The entire sixth row of the coded design matrix, the stay-at-home alternative, consists of zeros. The results are as follows:

```
                              Vacation Example


                                               Alaska, Alaska, Alaska,
 Hawaii Alaska Mexico California Maine   999     1249    1499

    1      0      0        0        0      0        0       0
    0      1      0        0        0      1        0       0
    0      0      1        0        0      0        0       0
    0      0      0        1        0      0        0       0
    0      0      0        0        1      0        0       0
    0      0      0        0        0      0        0       0

California, California, California, Hawaii, Hawaii, Hawaii,
    999         1249         1499      999     1249    1499

     0            0            0         1        0       0
     0            0            0         0        0       0
     0            0            0         0        0       0
     0            1            0         0        0       0
     0            0            0         0        0       0
     0            0            0         0        0       0
```

```
Maine, Maine, Maine, Mexico, Mexico, Mexico, Alaska, Alaska,  Alaska,
  999   1249   1499  999       1249    1499   Beach    Lake  Mountains

   0      0      0      0       0       0       0       0        0
   0      0      0      0       0       0       0       0        1
   0      0      0      0       1       0       0       0        0
   0      0      0      0       0       0       0       0        0
   0      1      0      0       0       0       0       0        0
   0      0      0      0       0       0       0       0        0

California, California, California, Hawaii, Hawaii,  Hawaii,
   Beach        Lake     Mountains   Beach    Lake  Mountains

     0           0           0         1       0        0
     0           0           0         0       0        0
     0           0           0         0       0        0
     0           1           0         0       0        0
     0           0           0         0       0        0
     0           0           0         0       0        0

                                                  Alaska,
Maine, Maine,   Maine, Mexico, Mexico,  Mexico,    Bed &    Alaska,  Alaska,
 Beach  Lake  Mountains Beach    Lake  Mountains Breakfast   Cabin    Hotel

   0      0       0       0       0        0         0         0        0
   0      0       0       0       0        0         0         0        1
   0      0       0       1       0        0         0         0        0
   0      0       0       0       0        0         0         0        0
   0      0       1       0       0        0         0         0        0
   0      0       0       0       0        0         0         0        0

California,                             Hawaii,
   Bed &     California,  California,    Bed &   Hawaii, Hawaii,
 Breakfast      Cabin        Hotel     Breakfast  Cabin   Hotel

     0            0            0           0        1       0
     0            0            0           0        0       0
     0            0            0           0        0       0
     0            1            0           0        0       0
     0            0            0           0        0       0
     0            0            0           0        0       0
```

```
Maine,                  Mexico,
Bed &      Maine, Maine,   Bed &   Mexico, Mexico,
Breakfast  Cabin  Hotel Breakfast  Cabin   Hotel  Place

    0        0      0       0        0       0    Hawaii
    0        0      0       0        0       0    Alaska
    0        0      0       0        0       1    Mexico
    0        0      0       0        0       0    California
    0        1      0       0        0       0    Maine
    0        0      0       0        0       0

Price      Scene          Lodge         Subj    Set    Form    c

  999    Beach        Cabin               1       1       1     1
  999    Mountains    Hotel               1       1       1     2
 1249    Beach        Hotel               1       1       1     2
 1249    Lake         Cabin               1       1       1     2
 1249    Mountains    Cabin               1       1       1     2
    .            .                  .     1       1       1     2
```

Analysis proceeds by running PROC PHREG as follows:

```
proc phreg data=coded brief;
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;
```

The results are as follows:

```
                              Vacation Example

                            The PHREG Procedure

                             Model Information

                   Data Set                   WORK.CODED
                   Dependent Variable         c
                   Censoring Variable         c
                   Censoring Value(s)         2
                   Ties Handling              BRESLOW

                 Number of Observations Read       21600
                 Number of Observations Used       21600
```

```
           Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                      Number of        Number of          Chosen          Not
           Pattern      Choices     Alternatives    Alternatives       Chosen

                 1         3600                6               1            5

                                 Convergence Status

                      Convergence criterion (GCONV=1E-8) satisfied.

                                Model Fit Statistics

                                         Without           With
                            Criterion     Covariates     Covariates

                            -2 LOG L       12900.668       6239.870
                            AIC            12900.668       6309.870
                            SBC            12900.668       6526.474

                            Testing Global Null Hypothesis: BETA=0

               Test                    Chi-Square       DF     Pr > ChiSq

               Likelihood Ratio         6660.7982       35         <.0001
               Score                    6601.7928       35         <.0001
               Wald                     2448.1475       35         <.0001

                             Multinomial Logit Parameter Estimates

                                         Parameter    Standard
                                   DF     Estimate       Error  Chi-Square  Pr > ChiSq

    Hawaii                          1      3.49208     0.47222     54.6856      <.0001
    Alaska                          1      0.17527     0.67139      0.0682      0.7940
    Mexico                          1      2.93013     0.47932     37.3706      <.0001
    California                      1      2.17915     0.49725     19.2058      <.0001
    Maine                           1      1.27770     0.54587      5.4787      0.0192
    Alaska,  999                    1      4.02423     0.46534     74.7858      <.0001
    Alaska, 1249                    1      1.81200     0.49473     13.4147      0.0002
    Alaska, 1499                    0            0           .           .           .
    California,  999                1      3.38438     0.19965    287.3498      <.0001
    California, 1249                1      1.22372     0.22445     29.7251      <.0001
    California, 1499                0            0           .           .           .
    Hawaii,  999                    1      3.61016     0.14157    650.2879      <.0001
    Hawaii, 1249                    1      1.45415     0.13050    124.1725      <.0001
    Hawaii, 1499                    0            0           .           .           .
    Maine,  999                     1      3.80918     0.26060    213.6577      <.0001
    Maine, 1249                     1      1.53370     0.27050     32.1475      <.0001
    Maine, 1499                     0            0           .           .           .
```

```
Mexico,  999                    1      3.45924      0.15495    498.4209       <.0001
Mexico, 1249                    1      1.41406      0.15693     81.1907       <.0001
Mexico, 1499                    0            0            .           .            .
Alaska, Beach                   1      1.01542      0.21881     21.5355       <.0001
Alaska, Lake                    1      0.48168      0.22639      4.5269       0.0334
Alaska, Mountains               0            0            .           .            .
California, Beach               1      1.47681      0.15536     90.3528       <.0001
California, Lake                1      0.84358      0.16138     27.3244       <.0001
California, Mountains           0            0            .           .            .
Hawaii, Beach                   1      1.29573      0.12493    107.5692       <.0001
Hawaii, Lake                    1      0.61301      0.12299     24.8444       <.0001
Hawaii, Mountains               0            0            .           .            .
Maine, Beach                    1      1.59739      0.20874     58.5584       <.0001
Maine, Lake                     1      0.64984      0.20203     10.3468       0.0013
Maine, Mountains                0            0            .           .            .
Mexico, Beach                   1      1.26780      0.13744     85.0857       <.0001
Mexico, Lake                    1      0.67632      0.13589     24.7716       <.0001
Mexico, Mountains               0            0            .           .            .
Alaska, Bed & Breakfast         1      1.00195      0.18862     28.2169       <.0001
Alaska, Cabin                   1     -1.33747      0.31958     17.5146       <.0001
Alaska, Hotel                   0            0            .           .            .
California, Bed & Breakfast     1      0.67004      0.13195     25.7875       <.0001
California, Cabin               1     -1.50239      0.16734     80.6060       <.0001
California, Hotel               0            0            .           .            .
Hawaii, Bed & Breakfast         1      0.63585      0.11523     30.4508       <.0001
Hawaii, Cabin                   1     -1.41004      0.13462    109.7155       <.0001
Hawaii, Hotel                   0            0            .           .            .
Maine, Bed & Breakfast          1      0.58532      0.15999     13.3848       0.0003
Maine, Cabin                    1     -1.50967      0.22377     45.5166       <.0001
Maine, Hotel                    0            0            .           .            .
Mexico, Bed & Breakfast         1      0.54835      0.11802     21.5891       <.0001
Mexico, Cabin                   1     -1.40762      0.15033     87.6707       <.0001
Mexico, Hotel                   0            0            .           .            .
```

There are zero coefficients for the reference level. Do we need this more complicated model instead of the simpler model? To answer this, first look at the coefficients. Are they similar across different destinations? In this case, they seem to be. This suggests that the simpler model might be sufficient.

More formally, the two models can be statistically compared. You can test the null hypothesis that the two models are not significantly different by comparing their likelihoods. The difference between two $-2\log(\mathcal{L}_C)$'s (the number reported under 'With Covariates' in the output) has a chi-square distribution. We can get the *df* for the test by subtracting the two *df* for the two likelihoods. The difference $6257.752 - 6239.870 = 17.882$ is distributed χ^2 with $35 - 11 = 24$ *df* ($p < 0.80869$). This more complicated model does not account for significantly more variance than the simpler model.

Vacation Example and Artificial Data Generation

This example does not illustrate any new capabilities. Rather, it shows how to make artificial data to test your design and your code for processing and analyzing the real data before the data are collected. This example goes into more detail about how to create artificial data than the other artificial data examples in this chapter. The artificial data creation step assumes that the design is in the final choice design format (not in the linear arrangement that comes from the `%MktEx` macro).

You begin by making a design. This example is based on the vacation example on page 339. The goal is to make an alternative-specific design for vacation destinations. The choice design has a destination attribute along with three other attributes, accommodations, scenery, and price. The design can be constructed from a linear arrangement with the following factors:

Factor	Destination	Attribute	Levels
X1	Hawaii	Accommodations	Cabin, Bed & Breakfast, Hotel
X2	Alaska	Accommodations	Cabin, Bed & Breakfast, Hotel
X3	Mexico	Accommodations	Cabin, Bed & Breakfast, Hotel
X4	California	Accommodations	Cabin, Bed & Breakfast, Hotel
X5	Maine	Accommodations	Cabin, Bed & Breakfast, Hotel
X6	Hawaii	Scenery	Mountains, Lake, Beach
X7	Alaska	Scenery	Mountains, Lake, Beach
X8	Mexico	Scenery	Mountains, Lake, Beach
X9	California	Scenery	Mountains, Lake, Beach
X10	Maine	Scenery	Mountains, Lake, Beach
X11	Hawaii	Price	\$999, \$1249, \$1499
X12	Alaska	Price	\$999, \$1249, \$1499
X13	Mexico	Price	\$999, \$1249, \$1499
X14	California	Price	\$999, \$1249, \$1499
X15	Maine	Price	\$999, \$1249, \$1499

The design additionally has a constant “stay at home” alternative.

The `%MktEx` macro can be used as follows to make a linear arrangement in 36 runs, which becomes a choice design with 36 choice sets:

```
%mktex(3 ** 15,                    /* 15 three-level factors            */
       n=36,                       /* 36 rows in linear arrang - 36 ch sets*/
       seed=205)                   /* random number seed                */
```

The linear arrangement is blocked into two blocks of size 18 as follows:

```
%mktblock(data=randomized,         /* block randomized design           */
          nblocks=2,               /* create two blocks of 18 choice sets */
          out=blocked,             /* output data set for blocked design  */
          seed=114)                /* random number seed                */
```

The rules for creating a choice design from a linear arrangement are stored in the following data set:

```
                                    /* make Place from the destinations     */
                                    /* make Lodge from x1-x5                */
                                    /* make Scene from x6-x10               */
                                    /* make Price from x11-x15              */
                                    /* make 'stay at home' from all missing */
data key;
   input Place $ 1-10 (Lodge Scene Price) ($);
   datalines;
Hawaii      x1  x6   x11
Alaska      x2  x7   x12
Mexico      x3  x8   x13
California  x4  x9   x14
Maine       x5  x10  x15
.           .   .    .
;
```

The following step creates the choice design and stores it in a permanent SAS data set:

```
%mktroll(design=blocked,            /* make choice design from blocked      */
                                    /* linear arrangement from %mktblock    */
         key=key,                   /* use rules in KEY data set            */
         alt=place,                 /* alternative name variable is Place   */
         out=sasuser.ChoiceDesign,  /* permanent data set for results       */
         options=nowarn,            /* don't warn about extra variables     */
         keep=block)                /* keep the blocking variable           */
```

There are many other ways you could make a choice design for this problem. You could also search a candidate set of choice sets or alternatives with the `%ChoicEff` macro. For this example, it does not matter. You can generate artificial data for a choice design stored in choice design format (one row per alternative) no matter how it was created.

The following steps assign formats to the levels of the attributes:

```
proc format;                        /* map 1, 2, 3 levels to actual levels  */
   value price 1 = '1499'      2 = '1749'            3 = '1999'  . = ' ';
   value scene 1 = 'Mountains' 2 = 'Lake'            3 = 'Beach' . = ' ';
   value lodge 1 = 'Cabin'     2 = 'Bed & Breakfast' 3 = 'Hotel' . = ' ';
   run;

data sasuser.ChoiceDesign;          /* assign formats to vars in the design */
   set sasuser.ChoiceDesign;
   format scene scene. lodge lodge. price price.;
   run;
```

The choice design data set contains the variable `Place`, which is a character variable with levels: 'Hawaii', 'Alaska', 'Mexico', 'California', 'Maine', and ' ' (for stay at home). The variables `Lodge`, `Scene`, and `Price` are numeric variables with values 1, 2, and 3 and with formatted values that are more descriptive.

The following step displays the choice design:

```
   proc print data=sasuser.ChoiceDesign; /* display sets and check results     */
      by block set; id block set;
      run;
```

It is important to display the choice design and ensure that it is correct. Some of the results are as follows:

```
          Block     Set     Place         Lodge              Scene         Price

            1         1     Hawaii        Cabin              Beach         1999
                            Alaska        Hotel              Lake          1999
                            Mexico        Cabin              Lake          1499
                            California    Cabin              Lake          1749
                            Maine         Hotel              Beach         1999


            1         2     Hawaii        Cabin              Mountains     1499
                            Alaska        Bed & Breakfast    Mountains     1999
                            Mexico        Bed & Breakfast    Mountains     1999
                            California    Hotel              Lake          1749
                            Maine         Hotel              Mountains     1749
            .
            .
            .
            1        18     Hawaii        Bed & Breakfast    Lake          1999
                            Alaska        Hotel              Beach         1749
                            Mexico        Cabin              Lake          1999
                            California    Bed & Breakfast    Beach         1749
                            Maine         Cabin              Lake          1749

            .
            .
            .
            2        36     Hawaii        Bed & Breakfast    Lake          1499
                            Alaska        Bed & Breakfast    Lake          1999
                            Mexico        Hotel              Mountains     1749
                            California    Cabin              Beach         1499
                            Maine         Cabin              Beach         1999
```

The following step evaluates the choice design by using the `%ChoicEff` macro:

```
                                   /* Evaluate the choice design          */
   %choiceff(data=sasuser.ChoiceDesign,/* candidate set of choice sets    */
            init=sasuser.ChoiceDesign(keep=set), /* select these sets     */
            intiter=0,             /* evaluate without internal iterations */
                                   /* alternative-specific effects model   */
                                   /* zero=none - no ref levels for place  */
                                   /* order=data - do not sort levels      */
            model=class(place / zero=none order=data)
                                   /* zero=' ' - no ref level for first    */
                                   /* factor (place), ordinary ref levels  */
                                   /* for other factors (price -- lodge).  */
                                   /* order=formatted - sort levels        */
                                   /* use blank sep to build interact terms*/
                  class(place * price place * scene place * lodge /
                        zero=' ' order=formatted separators='' ' ') /
                                   /* no ref level for place               */
                                   /* use blank sep to build interact terms*/
                  lprefix=0        /* lpr=0 labels created from just levels*/
                  cprefix=0,       /* cpr=0 names created from just levels */
            nsets=72,              /* number of choice sets                */
            nalts=6,               /* number of alternatives               */
            beta=zero)             /* assumed beta vector, Ho: b=0         */
```

The list of parameters, variances, and standard errors are as follows:

```
                                                                     Standard
      n Variable Name             Label                      Variance DF    Error

      1 Hawaii                    Hawaii                      0.76667  1  0.87560
      2 Alaska                    Alaska                      0.76667  1  0.87560
      3 Mexico                    Mexico                      0.76717  1  0.87588
      4 California                California                  0.86667  1  0.93095
      5 Maine                     Maine                       0.76717  1  0.87588
      6 Alaska_1499               Alaska 1499                 0.60000  1  0.77460
      7 Alaska_1749               Alaska 1749                 0.60000  1  0.77460
      8 California_1499           California 1499             0.80000  1  0.89443
      9 California_1749           California 1749             0.60000  1  0.77460
     10 Hawaii_1499               Hawaii 1499                 0.60000  1  0.77460
     11 Hawaii_1749               Hawaii 1749                 0.60000  1  0.77460
     12 Maine_1499                Maine 1499                  0.60150  1  0.77557
     13 Maine_1749                Maine 1749                  0.60150  1  0.77557
     14 Mexico_1499               Mexico 1499                 0.60150  1  0.77557
     15 Mexico_1749               Mexico 1749                 0.60150  1  0.77557
     16 AlaskaBeach               Alaska Beach                0.60000  1  0.77460
     17 AlaskaLake                Alaska Lake                 0.60000  1  0.77460
     18 CaliforniaBeach           California Beach            0.60000  1  0.77460
     19 CaliforniaLake            California Lake             0.60000  1  0.77460
```

```
      20 HawaiiBeach               Hawaii Beach                0.60000  1  0.77460
      21 HawaiiLake                Hawaii Lake                 0.60000  1  0.77460
      22 MaineBeach                Maine Beach                 0.60000  1  0.77460
      23 MaineLake                 Maine Lake                  0.60000  1  0.77460
      24 MexicoBeach               Mexico Beach                0.60000  1  0.77460
      25 MexicoLake                Mexico Lake                 0.60000  1  0.77460
      26 AlaskaBed___Breakfast     Alaska Bed & Breakfast      0.60000  1  0.77460
      27 AlaskaCabin               Alaska Cabin                0.60000  1  0.77460
      28 CaliforniaBed___Breakfast California Bed & Breakfast  0.60000  1  0.77460
      29 CaliforniaCabin           California Cabin            0.60000  1  0.77460
      30 HawaiiBed___Breakfast     Hawaii Bed & Breakfast      0.60000  1  0.77460
      31 HawaiiCabin               Hawaii Cabin                0.60000  1  0.77460
      32 MaineBed___Breakfast      Maine Bed & Breakfast       0.60000  1  0.77460
      33 MaineCabin                Maine Cabin                 0.60000  1  0.77460
      34 MexicoBed___Breakfast     Mexico Bed & Breakfast      0.60000  1  0.77460
      35 MexicoCabin               Mexico Cabin                0.60000  1  0.77460
                                                                       ==
                                                                       35
```

The next step, now that the choice design has been constructed and evaluated with the `%ChoicEff` macro, is making artificial data for some fictitious subjects. Before you do that, however, you should study the next three examples which explain some of the inner workings of the SAS DATA step. This background makes the data generation program clearer. The first program is a standard DATA step that creates a copy of the choice design. It is not a model for what you will do. Rather, it provides a standard and familiar DATA step program, which you can compare to what you will actually do. The following step creates a copy of a data set:

```
   data Test;          /* Make one copy of the input SAS data set.           */
                       /* DATA step does automatic looping, it automatically   */
                       /* writes out the observation to a SAS data set, and it */
                       /* automatically stops when it hits the end of file on  */
                       /* the input SAS data set.  With 216 observations in    */
                       /* the input SAS data set, there are 216 passes through */
                       /* the DATA step, and since there is no OUTPUT          */
                       /* statement, each observation is automatically written */
                       /* to the output SAS data set by an implicit OUTPUT.    */
      set sasuser.ChoiceDesign;
      run;
```

In the preceding step, SAS does most of the work for you. You can use this type of DATA step when there is a one to one correspondence between the input and the output. The DATA step that makes the artificial data does not have this one to one correspondence.

The following step creates a copy of the design data set, but this time, you must program all of the looping yourself:

```
data Test;          /* Make one copy of the input SAS data set.  There is   */
                    /* one pass through this DATA step, and the DO loop     */
                    /* reads 216 observations from the input SAS data set.  */

                    /* Do the looping yourself.                             */
   do i = 1 to 216; /* 36 choice sets and 6 alternatives = 216 observations */

                    /* SET statement with POINT=i reads the ith             */
                    /* observation, and the variable i is automatically     */
                    /* dropped from the output SAS data set.                */
      set sasuser.ChoiceDesign point=i;

      output;       /* Write out each observation to a SAS data set.        */
                    /* Without this statement, an OUTPUT statement          */
                    /* implicitly appears at the end of the DATA step,      */
                    /* which would have written out only the last           */
                    /* observation if the STOP statement hadn't stopped the */
                    /* DATA step first.                                     */
      end;

   stop;            /* Must specify STOP since it never hits the end of the */
                    /* data file (it never attempts to read past the last   */
                    /* record).  Infinite loop without this statement.      */
   run;
```

This step uses one pass through the DATA step with 216 reads of the input data set. The previous step uses 216 passes through the DATA step to perform 216 reads of the input data set. You can use a variation on this looping approach to create two copies of the output data set. Now there is no longer a one-to-one correspondence between the input data set and the output data set. The following step makes two copies of the input data set:

```
data Test;          /* Make two copies of the input SAS data set.           */
                    /* There is one pass through this DATA step, the outer  */
                    /* DO loop creates two copies, and the inner DO loop    */
                    /* reads 216 observations from the input SAS data set   */
                    /* (twice due to the outer DO loop).                    */

   do Subject = 1 to 2;/* Two copies.                                       */

                    /* Do the looping yourself.                             */
      do i = 1 to 216; /* 36 choice sets and 6 alternatives = 216 obs       */

                    /* SET statement reads the ith observation, and the     */
                    /* variable i is automatically dropped from the output  */
                    /* SAS data set.                                        */
         set sasuser.ChoiceDesign point=i;
         output;    /* Write out each observation to a SAS data set.        */
                    /* Without this statement, an OUTPUT statement          */
                    /* implicitly appears at the end of the DATA step,      */
                    /* which would have written out only the last           */
                    /* observation if the STOP statement hadn't stopped the */
                    /* DATA step first.                                     */
         end;
      end;

   stop;            /* Must specify STOP since it never hits the end of the */
                    /* data file (it never attempts to read past the last   */
                    /* record).  Infinite loop without this statement.      */
   run;
```

When making artificial data, you will use an approach similar to this last approach. For each block, for each subject, for each choice set, and for each alternative, you will read the design. Since the design is used repeatedly (1/2 of the design is used for each subject), you will read the design multiple times and write out multiple data points. There is not a one-to-one correspondence between the input design and the output data.

The step that creates the artificial data illustrates handling design variables in three different ways. `Lodge` and `Scene` are numeric variables with values 1, 2, and 3. These will work nicely for indexing arrays of hypothesized part-worth utility values. `Price` is no different from `Lodge` and `Scene`, but since `Price` is quantitative, you can use its value directly in the utility function if you choose to do so. `Place`, is a character variable with levels: 'Hawaii', 'Alaska', 'Mexico', 'California', 'Maine', and ' ' (for stay at home). You will want to convert those character values to integers when you assign utilities for each of the levels. The following informat does this for you:

```
                    /* Make an informat that will convert the character     */
                    /* values of the Place variable into the integers       */
                    /* 1 - 5.                                               */
proc format;
   invalue plinf 'Hawaii' = 1 'Alaska' = 2 'Mexico' = 3
                 'California' = 4 'Maine' = 5;
   run;
```

The following step creates artificial data:

```
data _null_;
   /* DATA _NULL_ allows you to use the DATA step without creating an       */
   /* output SAS data set.  The result is a list of artificial data written */
   /* to the SAS log.                                                       */
   /*                                                                       */
   /* Specify a list of expected utilities for each level of each           */
   /* attribute in an array for each attribute.  Recall that the Price,     */
   /* Scene, and Lodge have actual values 1, 2, 3 that correspond to the    */
   /* formatted values shown here:                                          */
   /*                                                                       */
   /* value price 1 = '1499'      2 = '1749'            3 = '1999'          */
   /* value scene 1 = 'Mountains' 2 = 'Lake'            3 = 'Beach'         */
   /* value lodge 1 = 'Cabin'     2 = 'Bed & Breakfast' 3 = 'Hotel'         */
   /*                                                                       */
   /* In the design data set, actual values are stored not the formatted    */
   /* values.  However, Place has only actual values.                       */
   /*                                                                       */
   /* When Place = 'Hawaii',                                                */
   /* input(place, plinf.) = 1, and the utility, dests[1] = 5;              */
   /* when Place = 'Alaska',                                                */
   /* input(place, plinf.) = 2, and the utility, dests[2] = -1;             */
   /* and so on.                                                            */
   /*                                                                       */
   /* When Scene = 1, and the formatted value is 'Mountains',               */
   /* scenes[1] = -1; and so on.                                            */
   /*                                                                       */
   /* Price has values 1 to 3 and they can be used directly in the          */
   /* utility function (or in a way similar to the other attributes).       */
   /*                                                                       */
   /* _temporary_ is used when the goal is for an array name and an         */
   /* index to access a list of values, and no data set variables are       */
   /* created or needed.                                                    */
   /*                                                                       */
   /* Be careful that your array names and variable names do not            */
   /* conflict with variable names in the input design data set.            */

   array dests[5]    _temporary_ (5 -1 4 3 2);
   array scenes[3]   _temporary_ (-1 0 1);
   array lodging[3]  _temporary_ (0 3 2);

   array u[6];              /* An array to store the utility of each alt    */

   Subject = 0;             /* Subject number                               */

   do s = 1 to 100;         /* Create data for 100 subjects per block       */

      i = 0;                /* Observation number in the design data set    */
      do BlockNum = 1 to 2; /* Create data for 2 blocks of 100 subjects     */
         Subject + 1;       /* Subject number                               */
```

```
/* put /         - go to a new line                                    */
/* blocknum 3.   - write the block number in 3 columns                 */
/* +2            - skip two columns                                    */
/* subject  3.   - write the subject number in 3 columns               */
/* +2            - skip two columns                                    */
/* @@;           - hold the output line for the data yet to come       */
put / blocknum 3. +2 subject 3. +2 @@;

do SetNum = 1 to 18; /* Loop over the 18 sets in a block               */
   do Alt = 1 to 6;  /* Loop over the 6 alts in a set                  */

      i + 1;         /* same as i = i + 1; design index,               */
                     /* i = 1 to 216 (2 blocks x 18 sets x 6 alts) */

                     /* Read the ith observation of the choice         */
                     /* design.  Note that you are reading the         */
                     /* choice design not the linear arrangement.  */
                     /* Just read in the variables that you need.  */
      set sasuser.ChoiceDesign(keep=place--price) point=i;

      if place ne ' '/* process the destinations differently         */
         then do;    /* from the constant 'stay at home'               */
         p = input(place, plinf.); /* map place values to 1-5          */
         u[alt] = 1 +              /* add 1 just for not at home   */
                  dests[p] +       /* util for destination             */
                  scenes[scene] +  /* util for scenery                 */
                  lodging[lodge] - /* util for lodging                 */
                  price;           /* negative util for price          */

         if place = 'Hawaii' and   /* add in Hawaii/Beach              */
            scene = 3 then         /* interaction                      */
            u[alt] = u[alt] + 2;

         if place = 'Maine' and    /* add Maine on a lake in a         */
            scene = 2 and lodge = 1/* cabin interaction                */
            then u[alt] = u[alt] + 1;

         end;

      else u[alt] = 0;                  /* util for stay at home   */

      u[alt] = u[alt] + 3 * normal(17); /* add error to utils          */
                                        /* change '3' to change        */
                                        /* the magnitude of error  */

      /* Alternatively, you can create Type I Gumbel errors for        */
      /* some scaling parameter b as follows:                          */
      /* u[alt] = u[alt] + b * log(-log(1 - uniform(104)));            */
      end;
```

```
         /* at this point you have gathered u1-u6 for all 6 alts         */
         m = max(of u1-u6);             /* max util over alts          */

         /* which one had the maximum util? do fuzzy comparison        */
         if      abs(u1 - m) < 1e-4 then c = 1; /* alt 1 is chosen      */
         else if abs(u2 - m) < 1e-4 then c = 2; /* alt 2 is chosen      */
         else if abs(u3 - m) < 1e-4 then c = 3; /* alt 3 is chosen      */
         else if abs(u4 - m) < 1e-4 then c = 4; /* alt 4 is chosen      */
         else if abs(u5 - m) < 1e-4 then c = 5; /* alt 5 is chosen      */
         else                            c = 6; /* alt 6 is chosen (home)*/

         put +(-1) c @@; /* write number of chosen alt. Use '@@' to hold */
                         /* the line for the rest of the data in this    */
                         /* block for this subject.                      */
                         /* +(-1) skips forward -1 space, which is how   */
                         /* you move back one space                      */
         end;
      end;
   end;
stop;                    /* explicitly stop since no end of file is hit  */
run;
```

You can copy and paste the results, which are displayed in the SAS log file, into the following DATA step[†] to input the results as you might do so when it comes time to analyze the data:

```
data results;            /* Read the input data.  List input for block   */
                         /* and subject.  Formatted input (fields of     */
                         /* size 1) for choices.                         */
   input Block Subject (choose1-choose18) (1.);
   datalines;
  1    1 416434213415311535
  2    2 115411541451441151
  1    3 132455331434113144
  2    4 313313244121311314
  1    5 451143532541411214
  2    6 311131311414411511
  .
  .
  .
  1  197 331335333114313112
  2  198 111314251521211141
  1  199 151351211114311131
  2  200 311314311151143141
;
```

[†]Alternatively, you could use a FILE statement in the preceding DATA step to write the data to a file (for example, FILE 'mytestdata.dat'; after the ARRAY statements). Then you could use INFILE 'mytestdata.dat'; before the INPUT statement instead of DATALINES and in stream data after the INPUT statement. If you do this, you must ensure that you do not rerun this program after the data are collected and wipe out the data.

It is important to display the contents of the results data set to ensure that you read the input data correctly. The following step displays the data set:

```
proc print; run;                          /* make sure data match input        */
```

Some of the results are as follows:

```
                                                     c  c  c  c  c  c  c  c  c
              S  c  c  c  c  c  c  c  c  c  h  h  h  h  h  h  h  h  h
              u  h  h  h  h  h  h  h  h  h  o  o  o  o  o  o  o  o  o
          B   b  o  o  o  o  o  o  o  o  o  o  o  o  o  o  o  o  o  o
          l   j  o  o  o  o  o  o  o  o  o  s  s  s  s  s  s  s  s  s
      O   o   e  s  s  s  s  s  s  s  s  s  e  e  e  e  e  e  e  e  e
      b   c   c  e  e  e  e  e  e  e  e  e  1  1  1  1  1  1  1  1  1
      s   k   t  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5  6  7  8

      1   1   1  4  1  6  4  3  4  2  1  3  4  1  5  3  1  1  5  3  5
      2   2   2  1  1  5  4  1  1  5  4  1  4  5  1  4  4  1  1  5  1
      3   1   3  1  3  2  4  5  5  3  3  1  4  3  4  1  1  3  1  4  4
      4   2   4  3  1  3  3  1  3  2  4  4  1  2  1  3  1  1  3  1  4
      5   1   5  4  5  1  1  4  3  5  3  2  5  4  1  4  1  1  2  1  4
      6   2   6  3  1  1  1  3  1  3  1  1  4  1  4  4  1  1  5  1  1
      .
      .
      .
    199   1 199  1  5  1  3  5  1  2  1  1  1  1  4  3  1  1  1  3  1
    200   2 200  3  1  1  3  1  4  3  1  1  1  5  1  1  4  3  1  4  1
```

You should ensure that there are no unexpected missing values and that the right number of observations (in this case, 2 blocks times 100 subjects) are present.

The following step merges the data and the choice design:

```
%mktmerge(design=sasuser.ChoiceDesign, /* merge the design data set         */
          data=results,                /* with the results                  */
          out=res2,                    /* create output data set res2       */
          blocks=block,                /* name of blocking variable         */
          nsets=18,                    /* number of choice sets per block   */
          nalts=6,                     /* number of alternatives            */
          setvars=choose1-choose18)    /* variables with the choices        */
```

The following step displays part of the resulting data set:

```
proc print data=res2(obs=18);          /* display some data - sanity check  */
   id block subject set;
   by block subject set;
   run;
```

It is again important to look at the results before proceeding. The results are as follows:

```
Block   Subject   Set   Place        Lodge              Scene        Price   c

  1        1       1    Hawaii       Cabin              Beach        1999    2
                        Alaska       Hotel              Lake         1999    2
                        Mexico       Cabin              Lake         1499    2
                        California   Cabin              Lake         1749    1
                        Maine        Hotel              Beach        1999    2
                                                                             2

  1        1       2    Hawaii       Cabin              Mountains    1499    1
                        Alaska       Bed & Breakfast    Mountains    1999    2
                        Mexico       Bed & Breakfast    Mountains    1999    2
                        California   Hotel              Lake         1749    2
                        Maine        Hotel              Mountains    1749    2
                                                                             2

  1        1       3    Hawaii       Hotel              Beach        1749    2
                        Alaska       Bed & Breakfast    Beach        1749    2
                        Mexico       Cabin              Mountains    1499    2
                        California   Cabin              Beach        1499    2
                        Maine        Hotel              Mountains    1499    2
                                                                             1
```

The following step codes the attributes for analysis:

```
proc transreg                             /* use proc transreg to code        */
      data=res2                           /* name of data set to code         */
      design=5000                         /* code big designs in chunks       */
                                          /* code up to 5000 obs at a time    */
      nozeroconstant                      /* don't zero constant variables    */
      norestoremissing;                   /* zeros in coded vars, not missings */
   model class(place /                    /* code Place variable              */
               zero=none                  /* use all nonmissing levels        */
               order=data)                /* don't sort levels                */
         class(place * price              /* code other vars                  */
               place * scene
               place * lodge /
               zero=none                  /* use all nonmissing levels        */
                                          /* including 0 reference levels     */
               order=formatted            /* do sort levels by formatted values*/
               separators='' ' ') /       /* use blank separator in interacts */
         lprefix=0;                       /* make labels just from levels     */
   output out=coded                       /* output coded data set            */
        (drop=_type_ _name_ intercept);/* drop vars that you don't need     */
   id subject set block c;                /* add extra vars that you do need  */
   run;
```

The following step customizes the output from PROC PHREG for choice modeling:

```
%phchoice( on )                        /* customize output from PHREG for   */
                                       /* choice models                     */
```

The following step analyzes the artificial data:

```
proc phreg data=coded brief;           /* do analysis with a brief summary  */
                                       /* of strata (set, subject, block)   */
   model c*c(2) = &_trgind / ties=breslow; /* standard choice model         */
   strata subject set block;           /* ID variables who taken together   */
   run;                                /* identify each individual choice   */
                                       /* set                               */
```

Some of the results are as follows:

```
          Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                    Number of        Number of          Chosen          Not
         Pattern      Choices     Alternatives    Alternatives       Chosen

               1         3600                6               1            5
```

It is important to note that there were 3600 choices (36 choice sets and 100 subjects per set). Each set consists of 6 alternatives—one was chosen and five were not. Each of the 3600 subject and set combinations jointly defines a separate stratum in the analysis. Any other pattern of results for this design and data set indicates an error in data collection, entry, or processing. More is said about this table at the end of this example.

The parameter estimates are as follows:

```
                     Multinomial Logit Parameter Estimates

                                  Parameter    Standard
                            DF     Estimate       Error  Chi-Square  Pr > ChiSq

Hawaii                       1      2.29050     0.17219    176.9392      <.0001
Alaska                       1     -1.58858     0.49315     10.3766      0.0013
Mexico                       1      1.87521     0.18389    103.9896      <.0001
California                   1      1.32370     0.21482     37.9705      <.0001
Maine                        1      0.77912     0.23117     11.3595      0.0008
Alaska 1499                  1      1.64559     0.42064     15.3050      <.0001
Alaska 1749                  1      0.65152     0.44333      2.1597      0.1417
Alaska 1999                  0            0           .           .           .
California 1499              1      0.97060     0.14844     42.7531      <.0001
California 1749              1      0.50921     0.13822     13.5731      0.0002
California 1999              0            0           .           .           .
```

```
Hawaii 1499                   1      0.98873     0.09483    108.7181      <.0001
Hawaii 1749                   1      0.49605     0.09585     26.7826      <.0001
Hawaii 1999                   0            0           .           .           .
Maine 1499                    1      1.19476     0.15276     61.1707      <.0001
Maine 1749                    1      0.74813     0.16447     20.6919      <.0001
Maine 1999                    0            0           .           .           .
Mexico 1499                   1      0.91560     0.10707     73.1267      <.0001
Mexico 1749                   1      0.51216     0.11346     20.3759      <.0001
Mexico 1999                   0            0           .           .           .
Alaska Beach                  1      1.54354     0.39272     15.4479      <.0001
Alaska Lake                   1      1.27276     0.41947      9.2065      0.0024
Alaska Mountains              0            0           .           .           .
California Beach              1      1.08297     0.12645     73.3523      <.0001
California Lake               1      0.47855     0.13229     13.0850      0.0003
California Mountains          0            0           .           .           .
Hawaii Beach                  1      1.80453     0.09781    340.3990      <.0001
Hawaii Lake                   1      0.42488     0.09485     20.0664      <.0001
Hawaii Mountains              0            0           .           .           .
Maine Beach                   1      0.99093     0.14340     47.7530      <.0001
Maine Lake                    1      0.38796     0.16197      5.7372      0.0166
Maine Mountains               0            0           .           .           .
Mexico Beach                  1      1.02413     0.10838     89.2899      <.0001
Mexico Lake                   1      0.51537     0.11251     20.9839      <.0001
Mexico Mountains              0            0           .           .           .
Alaska Bed & Breakfast        1      0.40897     0.27771      2.1687      0.1408
Alaska Cabin                  1     -2.43166     0.62578     15.0992      0.0001
Alaska Hotel                  0            0           .           .           .
California Bed & Breakfast    1      0.54281     0.10989     24.4010      <.0001
California Cabin              1     -0.80653     0.14042     32.9922      <.0001
California Hotel              0            0           .           .           .
Hawaii Bed & Breakfast        1      0.50693     0.09488     28.5458      <.0001
Hawaii Cabin                  1     -0.83312     0.09467     77.4467      <.0001
Hawaii Hotel                  0            0           .           .           .
Maine Bed & Breakfast         1      0.44868     0.12833     12.2231      0.0005
Maine Cabin                   1     -0.88742     0.17247     26.4743      <.0001
Maine Hotel                   0            0           .           .           .
Mexico Bed & Breakfast        1      0.42361     0.09781     18.7575      <.0001
Mexico Cabin                  1     -0.93363     0.11723     63.4222      <.0001
Mexico Hotel                  0            0           .           .           .
```

The parameter estimates for the destinations, including the stay at home (reference level) which is not displayed in the table are displayed in the following table along with the parameters that were used in the DATA step that made the artificial data:

Destination	Estimate	Parameter
Hawaii	2.29050	5
Alaska	-1.58858	-1
Mexico	1.87521	4
California	1.32370	3
Maine	0.77912	2
Stay at home	0	0

You would *not* expect these values to match, but you would expect a consistent ordering of the two, at least if you have generated enough data.

The price, scenery, and lodging parameters and estimates are as follows:

	Alaska	California	Hawaii	Maine	Mexico	Parameter
1499	1.64559	0.97060	0.98873	1.19476	0.91560	-1
1749	0.65152	0.50921	0.49605	0.74813	0.51216	-2
1999	0	0	0	0	0	-3
Beach	1.54354	1.08297	1.80453	0.99093	1.02413	1
Lake	1.27276	0.47855	0.42488	0.38796	0.51537	0
Mountains	0	0	0	0	0	-1
Bed & Breakfast	0.40897	0.54281	0.50693	0.44868	0.42361	3
Cabin	-2.43166	-0.80653	-0.83312	-0.88742	-0.93363	0
Hotel	0	0	0	0	0	2

All orderings are consistent. Furthermore, the parameter estimate for Hawaii at the beach is higher than the other beach parameter estimates. The parameter estimate for Maine in a cabin is not higher than the others even though you added one to the utility for that combination. However, in fact you added one for a three way interaction that included scenery, and that was not modeled.

You fit an alternative-specific effects model. That is, you fit a model with separate parameters for each destination. However, the data were created mostly assuming a main effects model. You could make the artificial data creation much more elaborate than this. Examples of two interaction terms were added. You could add many more. You could in fact have a completely different set of parameters for each destination. You could also have different parameters for different groups of individuals and concatenate the group results to make the final data set. However, the point of this exercise is not to add detailed realism to the artificial data generation. The point is just to have some data that you can use to ensure that you know how to read, process, code, and analyze the data before you collect them. The variances, covariances, and standard errors from the `%ChoicEff` macro show that all effects of interest are estimable and show other information about the design. Analyzing artificial data will not turn up anything beyond that. You could simply make random data to test your design.

The following step shows one way that you could create purely random data:

```
data _null_;
   do s = 1 to 100;
      do BlockNum = 1 to 2;
         Subject + 1;
         put / blocknum 3. +2 subject 3. +2 @@;
         do SetNum = 1 to 18;
            c = ceil(6 * uniform(17));
            put +(-1) c @@;
            end;
         end;
      end;
   stop;
   run;
```

The statement `c = ceil(6 * uniform(17))` generates random choices that are integers in the range 1 to 6. The expression `6 * uniform(17)` generates random values in the range 0 to 6, and the `ceil` function creates the ceiling, the integer greater than or equal to the value. This statement is the only statement that did not appear in the previous DATA _NULL_ step.

However you make the artificial data, particularly when you are new to choice modeling, creating and analyzing artificial data is a useful step to help ensure that you know what you are doing before you go to the time and expense of collecting data.

The last part of this example returns to the "Summary of Subjects, Sets, and Chosen and Unchosen Alternatives" table. Recall that in this example, it is as follows:

```
            Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                       Number of        Number of           Chosen          Not
           Pattern       Choices     Alternatives     Alternatives       Chosen

                 1          3600                6                1            5
```

The following PROC FREQ steps illustrate the logic that goes into making the first part of the table:

```
proc freq data=coded noprint;
   tables subject*set*block / list out=t1;
   run;

proc freq; tables count; run;
```

The number of times each stratum variable occurs in the data set is counted, and then the number of different frequencies is counted and displayed. The results are as follows:

```
                              The FREQ Procedure

                                Frequency Count

                                                   Cumulative    Cumulative
             COUNT    Frequency     Percent     Frequency      Percent
             -----------------------------------------------------------
                 6         3600      100.00          3600       100.00
```

3600 choice sets of size six occur in the data. The number of stratification variables that you use depends on the way you organize the input data. In this data set, subject number varies across blocks (the subject number varies from 1 to 200 rather than from 1 to 100 in block 1 and again from 1 to 100 in block 2). Hence, in this data set, you could drop the blocking variable from the STRATA statement and get the same results. The following steps illustrate:

```
proc freq data=coded noprint;
   tables subject*set / list out=t1;
   run;

proc freq; tables count; run;
```

The results match the PROC FREQ output from when the blocking variable was included.

Vacation Example with Alternative-Specific Attributes

This example discusses choosing the number of choice sets, designing the choice experiment, ensuring that certain key interactions are estimable, examining the design, blocking an existing design, evaluating the design, generating the questionnaire, generating artificial data, reading, processing, and analyzing the data, binary coding, generic attributes, alternative-specific effects, aggregating the data, analysis, and interpretation of the results. In this example, a researcher is interested in studying choice of vacation destinations. This page and the next page contain two summaries of the design, one with factors grouped by attribute and one grouped by destination.

This example is a modification of the previous example. Now, all alternatives do not have the same factors, and all factors do not have the same numbers of levels. There are still five destinations of interest: Hawaii, Alaska, Mexico, California, and Maine. Each alternative is composed of three factors like before: package cost, scenery, and accommodations, only now they do not all have the same levels, and the Hawaii and Mexico alternatives are composed of one additional attribute. For Hawaii and Alaska, the costs are \$1,249, \$1,499, and \$1,749; for California, the prices are \$999, \$1,249, \$1,499, and \$1,749; and for Mexico and Maine, the prices are \$999, \$1,249, and \$1,499. Scenery (mountains, lake, beach) and accommodations (cabin, bed & breakfast, and hotel) are the same as before. The Mexico trip now has the option of a side trip to sites of archaeological significance, via bus, for an additional cost of \$100. The Hawaii trip has the option of a side trip to an active volcano, via helicopter, for an additional cost of \$200. This is typical of the problems that marketing researchers face. We have many factors and *asymmetry*—each alternative is not composed of the same factors, and the factors do not all have the same numbers of levels.

Factor	Destination	Attribute	Levels
X1	Hawaii	Accommodations	Cabin, Bed & Breakfast, Hotel
X2	Alaska	Accommodations	Cabin, Bed & Breakfast, Hotel
X3	Mexico	Accommodations	Cabin, Bed & Breakfast, Hotel
X4	California	Accommodations	Cabin, Bed & Breakfast, Hotel
X5	Maine	Accommodations	Cabin, Bed & Breakfast, Hotel
X6	Hawaii	Scenery	Mountains, Lake, Beach
X7	Alaska	Scenery	Mountains, Lake, Beach
X8	Mexico	Scenery	Mountains, Lake, Beach
X9	California	Scenery	Mountains, Lake, Beach
X10	Maine	Scenery	Mountains, Lake, Beach
X11	Hawaii	Price	\$1249, \$1499, \$1749
X12	Alaska	Price	\$1249, \$1499, \$1749
X13	Mexico	Price	\$999, \$1249, \$1499
X14	California	Price	\$999, \$1249, \$1499, \$1749
X15	Maine	Price	\$999, \$1249, \$1499
X16	Hawaii	Side Trip	Yes, No
X17	Mexico	Side Trip	Yes, No

Factor	Destination	Attribute	Levels
X1	Hawaii	Accommodations	Cabin, Bed & Breakfast, Hotel
X6		Scenery	Mountains, Lake, Beach
X11		Price	\$1249, \$1499, \$1749
X16		Side Trip	Yes, No
X2	Alaska	Accommodations	Cabin, Bed & Breakfast, Hotel
X7		Scenery	Mountains, Lake, Beach
X12		Price	\$1249, \$1499, \$1749
X3	Mexico	Accommodations	Cabin, Bed & Breakfast, Hotel
X8		Scenery	Mountains, Lake, Beach
X13		Price	\$999, \$1249, \$1499
X17		Side Trip	Yes, No
X4	California	Accommodations	Cabin, Bed & Breakfast, Hotel
X9		Scenery	Mountains, Lake, Beach
X14		Price	\$999, \$1249, \$1499, \$1749
X5	Maine	Accommodations	Cabin, Bed & Breakfast, Hotel
X10		Scenery	Mountains, Lake, Beach
X15		Price	\$999, \$1249, \$1499

Choosing the Number of Choice Sets

We can use the `%MktRuns` autocall macro to suggest experimental design sizes. (All of the autocall macros used in this book are documented starting on page 803.) As before, we specify a list containing the number of levels of each factor as follows:

```
title 'Vacation Example with Asymmetry';

%mktruns(3 ** 14 4 2 2)
```

The results are as follows:

```
                    Vacation Example with Asymmetry

                            Design Summary

                        Number of
                        Levels       Frequency

                             2             2
                             3            14
                             4             1
```

```
                 Vacation Example with Asymmetry

     Saturated      = 34
     Full Factorial = 76,527,504

     Some Reasonable                         Cannot Be
        Design Sizes       Violations     Divided By

                72 *              0
               144 *              0
                36                2     8
               108                2     8
                54               18     4  8 12
                90               18     4  8 12
               126               18     4  8 12
                45               48     2  4  6  8 12
                63               48     2  4  6  8 12
                81               48     2  4  6  8 12
                34 S            151     3  4  6  8  9 12

 * - 100% Efficient design can be made with the MktEx macro.
 S - Saturated Design - The smallest design that can be made.
     Note that the saturated design is not one of the
     recommended designs for this problem.  It is shown
     to provide some context for the recommended sizes.

                 Vacation Example with Asymmetry

  n                    Design                           Reference

 72    2 ** 20  3 ** 24   4 **  1               Orthogonal Array
 72    2 ** 19  3 ** 20   4 **  1   6 **  1     Orthogonal Array
 72    2 ** 18  3 ** 16   4 **  1   6 **  2     Orthogonal Array
 72    2 ** 13  3 ** 25   4 **  1               Orthogonal Array
 72    2 ** 12  3 ** 21   4 **  1   6 **  1     Orthogonal Array
 72    2 ** 11  3 ** 24   4 **  1   6 **  1     Orthogonal Array
 72    2 ** 11  3 ** 17   4 **  1   6 **  2     Orthogonal Array
 72    2 ** 10  3 ** 20   4 **  1   6 **  2     Orthogonal Array
 72    2 **  9  3 ** 16   4 **  1   6 **  3     Orthogonal Array
144    2 ** 92  3 ** 24   4 **  1               Orthogonal Array
144    2 ** 91  3 ** 20   4 **  1   6 **  1     Orthogonal Array
144    2 ** 90  3 ** 16   4 **  1   6 **  2     Orthogonal Array
144    2 ** 85  3 ** 25   4 **  1               Orthogonal Array
144    2 ** 84  3 ** 21   4 **  1   6 **  1     Orthogonal Array
144    2 ** 83  3 ** 24   4 **  1   6 **  1     Orthogonal Array
144    2 ** 83  3 ** 17   4 **  1   6 **  2     Orthogonal Array
144    2 ** 82  3 ** 20   4 **  1   6 **  2     Orthogonal Array
144    2 ** 81  3 ** 16   4 **  1   6 **  3     Orthogonal Array
```

```
.
.
.
```

The output tells us the size of the saturated design, which is the number of parameters in the linear arrangement, and suggests design sizes. We need at least 34 choice sets, as shown by `(Saturated=34)` in the listing. Any size that is a multiple of 72 is optimal. We would recommend 72 choice sets, four blocks of size 18. However, like the previous vacation example, we use fewer choice sets so that we can illustrate getting an efficient but nonorthogonal design. A design with 36 choice sets is pretty good. Thirty-six is not divisible by $8 = 2 \times 4$, so we cannot have equal frequencies in the California price and Mexico and Hawaii side trip combinations. This should not pose any problem. This leaves only 2 error *df* for the linear model, but in the choice model, we have adequate error *df*.

Designing the Choice Experiment

This problem requires a design with 1 four-level factor for price and 4 three-level factors for price. There are 10 three-level factors for scenery and accommodations as before. Also, we need 2 two-level factors for the two side trips. Note that we do not need a factor for the price or mode of transportation of the side trips since they are constant within each trip. With the `%MktEx` macro, making an asymmetric design is no more difficult than making a symmetric design. The following step creates the design:[*]

```
%mktex(3 ** 13 4 3 2 2, n=36, seed=205)

%mkteval(data=randomized)
```

The last part of the results are as follows:

```
                      Vacation Example with Asymmetry

                            The OPTEX Procedure

                          Class Level Information

                          Class  Levels  Values

                          x1        3    1 2 3
                          x2        3    1 2 3
                          x3        3    1 2 3
                          x4        3    1 2 3
                          x5        3    1 2 3
                          x6        3    1 2 3
                          x7        3    1 2 3
                          x8        3    1 2 3
```

[*]Due to machine, SAS release, and macro differences, you might not get exactly the same design used in this book, but the differences should be slight.

```
                                        x9         3     1 2 3
                                        x10        3     1 2 3
                                        x11        3     1 2 3
                                        x12        3     1 2 3
                                        x13        3     1 2 3
                                        x14        4     1 2 3 4
                                        x15        3     1 2 3
                                        x16        2     1 2
                                        x17        2     1 2

                                   Vacation Example with Asymmetry

                                                                                    Average
                                                                                   Prediction
        Design                                                                      Standard
        Number     D-Efficiency     A-Efficiency     G-Efficiency       Error
        ------------------------------------------------------------------------
             1         98.8874           97.5943           97.4925          0.9718

                                 Vacation Example with Asymmetry
                            Canonical Correlations Between the Factors
                       There are 2 Canonical Correlations Greater Than 0.316

        x1  x2  x3  x4  x5  x6  x7  x8  x9  x10 x11 x12 x13  x14  x15  x16  x17

    x1  1   0   0   0   0   0   0   0   0   0   0   0   0    0    0    0    0
    x2  0   1   0   0   0   0   0   0   0   0   0   0   0    0    0    0    0
    x3  0   0   1   0   0   0   0   0   0   0   0   0   0    0    0    0    0
    x4  0   0   0   1   0   0   0   0   0   0   0   0   0    0    0    0    0
    x5  0   0   0   0   1   0   0   0   0   0   0   0   0    0    0    0    0
    x6  0   0   0   0   0   1   0   0   0   0   0   0   0    0    0    0    0
    x7  0   0   0   0   0   0   1   0   0   0   0   0   0    0    0    0    0
    x8  0   0   0   0   0   0   0   1   0   0   0   0   0    0    0    0    0
    x9  0   0   0   0   0   0   0   0   1   0   0   0   0    0    0    0    0
    x10 0   0   0   0   0   0   0   0   0   1   0   0   0    0    0    0    0
    x11 0   0   0   0   0   0   0   0   0   0   1   0   0    0    0    0    0
    x12 0   0   0   0   0   0   0   0   0   0   0   1   0    0    0    0    0
    x13 0   0   0   0   0   0   0   0   0   0   0   0   1    0    0.25 0    0
    x14 0   0   0   0   0   0   0   0   0   0   0   0   0    1    0    0.33 0.33
    x15 0   0   0   0   0   0   0   0   0   0   0   0   0.25 0    1    0    0
    x16 0   0   0   0   0   0   0   0   0   0   0   0   0    0.33 0    1    0
    x17 0   0   0   0   0   0   0   0   0   0   0   0   0    0.33 0    0    1
```

The macro found a very nice, almost orthogonal and almost 99% *D*-efficient design in 40 seconds. However, we will not use this design. Instead, we will make a larger design with interactions.

Ensuring that Certain Key Interactions are Estimable

Next, we ensure that certain key interactions are estimable. Specifically, it is good if in the aggregate, the interactions between price and accommodations are estimable for each destination. We would like the following interactions to be estimable: `x1*x11 x2*x12 x3*x13 x4*x15 x5*x15`. We use the `%MktEx` macro as follows:

```
%mktex(3 ** 13 4 3 2 2, n=36, seed=205
       interact=x1*x11 x2*x12 x3*x13 x4*x14 x5*x15)
```

The macro immediately displays the following errors:

```
ERROR: More parameters than runs.
       If you really want to do this, specify RIDGE=.
       There are 36 runs with 56 parameters.
ERROR: The MKTEX macro ended abnormally.
```

If we want interactions to be estimable, we need more choice sets. The number of parameters is 1 for the intercept, $14\times(3-1)+(4-1)+2\times(2-1) = 33$ for main effects, and $4\times(3-1)\times(3-1)+(4-1)\times(3-1) = 22$ for interactions for a total of $1 + 33 + 22 = 56$ parameters. This means we need at least 56 choice sets, and ideally for this design with 2, 3, and 4 level factors, we would like the number of sets to be divisible by 2×2, 2×3, 2×4, 3×3, and 3×4. Sixty is divisible by 2, 3, 4, 6, and 12 so is a reasonable design size. Sixty choice sets could be divided into three blocks of size 20, four blocks of size 15, or five blocks of size 12. Seventy-two choice sets is better, since unlike 60, 72 can be divided by 9. Unfortunately, 72 would require one more block.

We can use the `%MktRuns` macro to help us choose the number of choice sets. We also specified a keyword option `max=` to consider only the 45 design sizes from the minimum of 56 up to 100 as follows:

```
title 'Vacation Example with Asymmetry';
%mktruns(3 ** 13 4 3 2 2, interact=x1*x11 x2*x12 x3*x13 x4*x14 x5*x15, max=45)
```

The results are as follows:

```
                    Vacation Example with Asymmetry

                            Design Summary

                        Number of
                        Levels       Frequency

                             2            2
                             3           14
                             4            1

                    Vacation Example with Asymmetry

   Saturated      = 56
   Full Factorial = 76,527,504
```

```
     Some Reasonable                Cannot Be
        Design Sizes   Violations   Divided By

                  72           58   27  81 108
                  81           79    2   4   6   8  12  18  24  36 108
                  90           95    4   8  12  24  27  36  81 108
                  63          133    2   4   6   8  12  18  24  27  36  81 108
                  99          133    2   4   6   8  12  18  24  27  36  81 108
                  96          174    9  18  27  36  81 108
                  60          178    8   9  18  24  27  36  81 108
                  84          178    8   9  18  24  27  36  81 108
                  66          194    4   8   9  12  18  24  27  36  81 108
                  78          194    4   8   9  12  18  24  27  36  81 108
                  56 S        232    3   6   9  12  18  24  27  36  81 108

            S - Saturated Design - The smallest design that can be made.
                Note that the saturated design is not one of the
                recommended designs for this problem.  It is shown
                to provide some context for the recommended sizes.
```

We see that 72 cannot be divided by $27 = 9 \times 3$ so, for example, the Maine accommodation/price combinations cannot occur with equal frequency with each of the three-level factors. We see that 72 cannot be divided by $81 = 9 \times 9$ so, for example, the Mexico accommodation/price combinations cannot occur with equal frequency with each of the Hawaii accommodation/price combinations. We see that 72 cannot be divided by $108 = 9 \times 12$ so, for example, the California accommodation/price combinations cannot occur with equal frequency with each of the Maine accommodation/price combinations. With interactions, there are many higher-order opportunities for nonorthogonality. However, usually we are not be overly concerned about potential unequal cell frequencies on combinations of attributes in different alternatives.

The smallest number of runs in the table is 60. While 72 is better in that it can be divided by more numbers, either 72 or 60 should work fine. We pick the larger number and run the `%MktEx` macro again with `n=72` specified as follows:

```
%mktex(3 ** 13 4 3 2 2, n=72, seed=368,
       interact=x1*x11 x2*x12 x3*x13 x4*x14 x5*x15)
```

The final *D*-efficiency table is as follows:

```
                        Vacation Example with Asymmetry
                                                                    Average
                                                                   Prediction
       Design                                                       Standard
       Number     D-Efficiency     A-Efficiency     G-Efficiency      Error
       ------------------------------------------------------------------------
            1         89.8309          79.7751          94.4393         0.8819
```

Sometimes, particularly in models with two-way interactions, we can do better by having `%MktEx` do pairwise exchanges in the coordinate-exchange algorithm instead of working on a single column at a time. You can always specify `exchange=2` and `order=sequential` to get all possible pairs, but this can be very time consuming. Alternatively, you can use the `order=matrix=`*SAS-data-set* option and tell `%MktEx` exactly which pairs of columns to work on. That approach is illustrated in the following steps:

```
data mat;
   do a = 1 to 17;
      b = .;
      output;
      end;
   do a = 1 to 5;
      b = 10 + a;
      output;
      end;
   run;

proc print; run;

%mktex(3 ** 13 4 3 2 2,             /* all attrs of all alternatives        */
       n=72,                        /* number of choice sets                */
       seed=368,                    /* random number seed                   */
       order=matrix=mat,            /* identifies pairs of cols to work on  */
       interact=x1*x11 x2*x12 x3*x13 x4*x14 x5*x15) /* interactions         */
```

The data set `Mat` is as follows:

```
                        Vacation Example with Asymmetry

                                Obs     a     b

                                  1     1     .
                                  2     2     .
                                  3     3     .
                                  4     4     .
                                  5     5     .
                                  6     6     .
                                  7     7     .
                                  8     8     .
                                  9     9     .
                                 10    10     .
                                 11    11     .
                                 12    12     .
                                 13    13     .
                                 14    14     .
                                 15    15     .
                                 16    16     .
                                 17    17     .
```

```
18     1     11
19     2     12
20     3     13
21     4     14
22     5     15
```

It has two columns. The values in the data set indicate the pairs of columns that `%MktEx` should work on together. Missing values are replaced by a random column number for every row and for every pass through the design. This data set instructs `%MktEx` to sequentially go through every column each time paired with some other random column, then work through all of the interaction pairs, `x1*x11`, `x2*x12`, and so on. This performs 22 pairwise exchanges in every row, which is many fewer exchanges than the $17 \times 16/2 = 136$ that are required with `exchange=2` and `order=sequential`. There are many other combinations that you might consider. A few examples are as follows:

One		Two		Three		Four		Five		
1	.	1	11	1	1	1	11	1	.	.
2	.	2	12	2	2	2	12	2	.	.
3	.	3	13	3	3	3	13	3	.	.
4	.	4	14	4	4	4	14	4	.	.
5	.	5	15	5	5	5	15	5	.	.
6	.	6	.	6	6	6	6	6	.	.
7	.	7	.	7	7	7	7	7	.	.
8	.	8	.	8	8	8	8	8	.	.
9	.	9	.	9	9	9	9	9	.	.
10	.	10	.	10	10	10	10	10	.	.
11	.			11	11			11	.	.
12	.			12	12			12	.	.
13	.			13	13			13	.	.
14	.			14	14			14	.	.
15	.			15	15			15	.	.
16	.			16	16			16	.	.
17	.			17	17			17	.	.
1	11			1	11			1	11	.
2	12			2	12			2	12	.
3	13			3	13			3	13	.
4	14			4	14			4	14	.
5	15			5	15			5	15	.

Set one is the set we just used. Each column is paired with a random column and every interaction pair is mentioned. Set two is like set one except it consists of only 10 pairs and the interaction columns are only paired with the other columns in its interaction term. Set three names each factor twice and then has the usual pairs for interactions. This requests 17 single-column exchanges followed by 5 pairwise exchanges. When a column is repeated, all but the first instance is ignored. `%MktEx` does not consider all pairs of a factor with itself. Set four is similar to set 3 but the interaction columns are only paired with the other columns in its interaction term. Set five is like set one except three-way exchanges are performed and a random column is added to each exchange. There are many other possibilities. It is impossible to know what will work best, but often, expending some effort to consider exchanges in pairs for two-way interactions or in triples for three-way interactions is rewarded with a small gain in

D-efficiency.

The macro displays the following notes:

```
NOTE: Generating the candidate set.
NOTE: Performing 20 searches of 243 candidates, full-factorial=76,527,504.
NOTE: Generating the orthogonal array design, n=72.
```

The candidate-set search is using a fractional-factorial candidate set with $3^5 = 243$ candidates. The two-level factors in the candidate set are made from three-level factors by coding down. *Coding down* replaces an m-level factor with a factor with fewer than m levels, for example, a two-level factor could be created from a three-level factor: $((1\ 2\ 3) \Rightarrow (1\ 2\ 1))$. The four-level factor in the candidate set is made from 2 three-level factors and coding down. $((1\ 2\ 3)\times(1\ 2\ 3) \Rightarrow (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9) \Rightarrow (1\ 2\ 3\ 4\ 1\ 2\ 3\ 4\ 1))$. The orthogonal array used for the partial initialization in the coordinate-exchange steps has 72 runs. Some of the results are as follows:

```
                  Vacation Example with Asymmetry

                      Algorithm Search History

                              Current         Best
Design    Row,Col  D-Efficiency  D-Efficiency  Notes
----------------------------------------------------------
     1      Start       84.8774       84.8774  Can
     1        End       84.8774

     2      Start       41.1012                Tab
     2      8   5       84.9292       84.9292
     2     16   2       84.9450       84.9450
     2     17  15       85.0008       85.0008
     2     18   3       85.0423       85.0423
     .
     .
     .
     2     42  14       87.3823       87.3823
     2     66   5       87.4076       87.4076
     2      2  13       87.4113       87.4113
     2        End       87.4113
     .
     .
     .
    11      Start       41.1012                Tab
    11        End       87.2914
```

```
             12      Start       55.7719                    Ran,Mut,Ann
             12     53  16       87.4195       87.4195
             12     48   9       87.4355       87.4355
             12     49   6       87.4688       87.4688
             12     50   1       87.4688       87.4688
              .
              .
              .
             12      9   4       90.3157       90.3157
             12        End       90.3157
              .
              .
              .
             17      Start       58.3436                    Ran,Mut,Ann
             17        End       90.5685

NOTE: Quitting the algorithm search step after 10.03 minutes and 17 designs.

                          Design Search History

                              Current          Best
          Design    Row,Col  D-Efficiency  D-Efficiency  Notes
          ----------------------------------------------------------
               0    Initial       90.7877       90.7877  Ini

               1      Start       59.2298                Ran,Mut,Ann
               1        End       90.3158
               .
               .
               .
              14      Start       58.8515                Ran,Mut,Ann
              14        End       90.3433

NOTE: Quitting the design search step after 20.17 minutes and 14 designs.


                     Vacation Example with Asymmetry

                        Design Refinement History

                              Current          Best
          Design    Row,Col  D-Efficiency  D-Efficiency  Notes
          ----------------------------------------------------------
               0    Initial       90.7877       90.7877  Ini

               1      Start       88.9486                Pre,Mut,Ann
               1        End       90.6106
```

```
                      2      Start       87.6249                 Pre,Mut,Ann
                      2     64   4       90.7886       90.7886
                      2        End       90.7803
                      .
                      .
                      .
                      5      Start       89.3771                 Pre,Mut,Ann
                      5        End       90.5049

    NOTE: Quitting the refinement step after 5.60 minutes and 5 designs.

                               Vacation Example with Asymmetry

                                     The OPTEX Procedure

                                   Class Level Information

                                    Class  Levels  Values

                                    x1       3     1 2 3
                                    x2       3     1 2 3
                                    x3       3     1 2 3
                                    x4       3     1 2 3
                                    x5       3     1 2 3
                                    x6       3     1 2 3
                                    x7       3     1 2 3
                                    x8       3     1 2 3
                                    x9       3     1 2 3
                                    x10      3     1 2 3
                                    x11      3     1 2 3
                                    x12      3     1 2 3
                                    x13      3     1 2 3
                                    x14      4     1 2 3 4
                                    x15      3     1 2 3
                                    x16      2     1 2
                                    x17      2     1 2

                                                                       Average
                                                                      Prediction
        Design                                                         Standard
        Number     D-Efficiency     A-Efficiency     G-Efficiency        Error
        ------------------------------------------------------------------------
             1         90.7886          81.5474          93.3553          0.8819
```

The macro ran in approximately 36 minutes. The algorithm search history shows that the candidate-set approach (`Can`) used in design 1 found a design that is 84.8774% *D*-efficient. The macro makes no attempt to improve this design, unless there are restrictions on the design, until the end in the design refinement step, and only if it is the best design found.

Designs 2 through 11 used the coordinate-exchange algorithm with an orthogonal array initialization (`Tab`). For this problem, the orthogonal array initialization initializes all 72 rows. For other problems, when the number of runs in the design is greater than the number of runs in the nearest orthogonal array, the remaining rows are randomly initialized. The orthogonal array initialization usually works very well when all but at most a very few rows and columns are left uninitialized and there are no interactions or restrictions. That is not the case in this problem, and when the algorithm switches to a fully-random initialization in design 12, it immediately does better.

The algorithm search phase picked the coordinate-exchange algorithm with a random initialization, random mutations, and simulated annealing as the algorithm to use in the next step, the design search step. The design search history is initialized with the best design (*D*-efficiency = 90.7877) found so far. The design search phase starts out with the initial design (`Ini`) found in the algorithm search phase. The macro finds a design with *D*-efficiency = 90.3433.

The final set of iterations tries to improve the best design found so far. Random mutations (`Ran`), simulated annealing (`Ann`), and level exchanges are used on the previous best (`Pre`) design. The random mutations are responsible for making the *D*-efficiency of the starting design worse than the previous best *D*-efficiency. In this case, the design refinement step found a very slight improvement on the best design found by the design search step.

Each stage ended before the maximum number of iterations and displayed a note. The macro displays the following notes:

```
NOTE: Quitting the algorithm search step after 10.03 minutes and 17 designs.
NOTE: Quitting the design search step after 20.17 minutes and 14 designs.
NOTE: Quitting the refinement step after 5.60 minutes and 5 designs.
```

The default values for `maxtime=10 20 5` constrain the three steps to run in an approximate maximum time of 10, 20, and 5 minutes. Fewer iterations are performed with `order=matrix` than with the default single-column exchanges because each pair of exchanges takes longer than a single exchange. For example, with two three-level factors, a pairwise exchange considers $3 \times 3 = 9$ exchanges, whereas a single exchange considers 3 exchanges. However, a single design with a random initialization and annealing, is faster and better than the full `%MktEx` run with a single-column exchange. This is requested by specifying `options=quickr` as follows:

```
data mat;
   do a = 1 to 17;
      b = .;
      output;
      end;
   do a = 1 to 5;
      b = 10 + a;
      output;
      end;
   run;

proc print; run;
```

```
%mktex(3 ** 13 4 3 2 2,            /* all attrs of all alternatives        */
       n=72,                       /* number of choice sets                */
       seed=368,                   /* random number seed                   */
       order=matrix=mat,           /* identifies pairs of cols to work on  */
               quickr              /* very quick run with random init      */
       interact=x1*x11 x2*x12 x3*x13 x4*x14 x5*x15) /* interactions        */
```

These steps are not run, and we will use the design created with the previous steps.

Examining the Design

We can use the `%MktEval` macro to evaluate the goodness of this design as follows:

```
%mkteval(data=design)
```

Some of the results are as follows:

```
                         Vacation Example with Asymmetry
                    Canonical Correlations Between the Factors
               There are 0 Canonical Correlations Greater Than 0.316

             x1      x2      x3      x4      x5      x6      x7      x8      x9

      x1     1       0.11    0.09    0.17    0.14    0.03    0.09    0.11    0.09
      x2     0.11    1       0.18    0.13    0.09    0.14    0.09    0.07    0.15
      x3     0.09    0.18    1       0.20    0.11    0.09    0.09    0.13    0.09
      x4     0.17    0.13    0.20    1       0.13    0.11    0.07    0.09    0.15
      x5     0.14    0.09    0.11    0.13    1       0.03    0.12    0.05    0.13
      x6     0.03    0.14    0.09    0.11    0.03    1       0.10    0.04    0.11
      x7     0.09    0.09    0.09    0.07    0.12    0.10    1       0.08    0.09
      x8     0.11    0.07    0.13    0.09    0.05    0.04    0.08    1       0.07
      x9     0.09    0.15    0.09    0.15    0.13    0.11    0.09    0.07    1
      .
      .
      .

                         Vacation Example with Asymmetry
                             Summary of Frequencies
               There are 0 Canonical Correlations Greater Than 0.316
                         * - Indicates Unequal Frequencies

                              Frequencies

               *    x1        25 24 23
               *    x2        24 25 23
                    x3        24 24 24
               *    x4        25 24 23
```

```
*    x5          25 23 24
*    x6          26 22 24
*    x7          22 26 24
*    x8          23 26 23
*    x9          23 27 22
*    x10         24 26 22
*    x11         23 24 25
*    x12         23 25 24
*    x13         25 23 24
*    x14         19 17 19 17
     x15         24 24 24
     x16         36 36
*    x17         37 35
*    x1 x2       7 9 9 8 8 8 9 8 6
*    x1 x3       9 9 7 7 8 9 8 7 8
*    x1 x4       7 9 9 11 7 6 7 8 8
*    x1 x5       8 8 9 10 8 6 7 7 9
*    x1 x6       9 8 8 9 7 8 8 7 8
*    x1 x7       8 10 7 6 9 9 8 7 8
*    x1 x8       9 8 8 8 8 8 6 10 7
*    x1 x9       7 9 9 8 9 7 8 9 6
*    x1 x10      9 8 8 8 10 6 7 8 8
*    x1 x11      8 8 9 7 8 9 8 8 7
*    x1 x12      7 8 10 9 8 7 7 9 7
*    x1 x13      9 9 7 7 8 9 9 6 8
*    x1 x14      7 6 7 5 7 4 6 7 5 7 6 5
*    x1 x15      10 7 8 8 8 8 6 9 8
*    x1 x16      12 13 12 12 12 11
*    x1 x17      12 13 14 10 11 12
.
.
.
*    x12 x13     6 9 8 10 6 9 9 8 7
*    x12 x14     5 5 6 7 8 6 6 5 6 6 7 5
*    x12 x15     7 9 7 9 7 9 8 8 8
*    x12 x16     12 11 13 12 11 13
*    x12 x17     12 11 13 12 12 12
.
.
.
     N-Way       1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

We can use the %MktEx macro to check the design and display the information matrix and variance matrix as follows:

```
%mktex(3 ** 13 4 3 2 2,            /* all attrs of all alternatives       */
       n=72,                       /* number of choice sets               */
       examine=i v,                /* show information & variance matrices */
       options=check,              /* check initial design efficiency     */
       init=randomized,            /* initial design                      */
       interact=x1*x11 x2*x12 x3*x13 x4*x14 x5*x15) /* interactions        */
```

A small part of the results are as follows:

```
                          Vacation Example with Asymmetry
                                Information Matrix

              Intercept x11     x12     x21     x22     x31     x32     x41

     Intercept   72.000    2.121  -1.225   2.121   1.225   0       0        2.121
     x11          2.121   73.500   0.866   1.500  -6.062  -6.000   0       -7.500
     x12         -1.225    0.866  70.500   0.866   4.500  -1.732  -3.000   -9.526
     x21          2.121    1.500   0.866  73.500  -0.866  -6.000   0       -3.000
     x22          1.225   -6.062   4.500  -0.866  70.500  -3.464 -12.000   1.732
     x31          0       -6.000  -1.732  -6.000  -3.464  72.000   0       -1.500
     x32          0        0      -3.000   0     -12.000   0      72.000    2.598
     x41          2.121   -7.500  -9.526  -3.000   1.732  -1.500   2.598  73.500
     .
     .
     .

                          Vacation Example with Asymmetry
                                 Variance Matrix

              Intercept x11     x12     x21     x22     x31     x32     x41

     Intercept    0.015   -0.001   0.000  -0.001  -0.001  -0.000  -0.000  -0.001
     x11         -0.001    0.017  -0.001   0.001   0.002   0.001   0.001   0.002
     x12          0.000   -0.001   0.017  -0.001  -0.001   0.001   0.000   0.002
     x21         -0.001    0.001  -0.001   0.017   0.000   0.002   0.000   0.001
     x22         -0.001    0.002  -0.001   0.000   0.018   0.001   0.003   0.000
     x31         -0.000    0.001   0.001   0.002   0.001   0.016   0.000  -0.000
     x32         -0.000    0.001   0.000   0.000   0.003   0.000   0.018  -0.001
     x41         -0.001    0.002   0.002   0.001   0.000  -0.000  -0.001   0.017
     .
     .
     .
```

Blocking an Existing Design

An existing design is blocked using the `%MktBlock` macro. The macro takes the observations in an existing design and optimally sorts them into blocks. Here, we are seeing how to block the linear version of the choice design, but the macro can also be used directly on the choice design. The following step blocks the design:

```
%mktblock(data=randomized, nblocks=4, out=sasuser.AsymVacLinDesBlckd, seed=114)
```

This step took 2 seconds. Some of the results including the one-way frequencies within blocks are as follows:

```
                         Vacation Example with Asymmetry
                     Canonical Correlations Between the Factors
                There are 0 Canonical Correlations Greater Than 0.316

             Block    x1      x2      x3      x4      x5      x6      x7      x8

     Block   1        0.08    0.06    0.10    0.06    0.08    0.08    0.08    0.06
     x1      0.08     1       0.11    0.09    0.17    0.14    0.03    0.09    0.11
     x2      0.06     0.11    1       0.18    0.13    0.09    0.14    0.09    0.07
     x3      0.10     0.09    0.18    1       0.20    0.11    0.09    0.09    0.13
     x4      0.06     0.17    0.13    0.20    1       0.13    0.11    0.07    0.09
     x5      0.08     0.14    0.09    0.11    0.13    1       0.03    0.12    0.05
     x6      0.08     0.03    0.14    0.09    0.11    0.03    1       0.10    0.04
     x7      0.08     0.09    0.09    0.09    0.07    0.12    0.10    1       0.08
     x8      0.06     0.11    0.07    0.13    0.09    0.05    0.04    0.08    1
     .
     .
     .

                         Vacation Example with Asymmetry
                             Summary of Frequencies
                There are 0 Canonical Correlations Greater Than 0.316
                         * - Indicates Unequal Frequencies

                                  Frequencies

                 Block        18 18 18 18
            *    x1           25 23 24
            *    x2           25 24 23
                 x3           24 24 24
            *    x4           25 24 23
            *    x5           25 24 23
            *    x6           22 26 24
```

```
*    x7           24 26 22
*    x8           26 23 23
*    x9           23 22 27
*    x10          24 26 22
*    x11          23 24 25
*    x12          24 23 25
*    x13          24 23 25
*    x14          17 19 19 17
     x15          24 24 24
     x16          36 36
*    x17          37 35
*    Block x1     6 6 6 6 5 7 6 6 6 7 6 5
*    Block x2     6 6 6 6 6 6 7 6 5 6 6 6
*    Block x3     6 6 6 6 6 6 6 7 5 6 5 7
*    Block x4     7 6 5 6 6 6 6 6 6 6 6 6
*    Block x5     6 6 6 7 5 6 6 7 5 6 6 6
*    Block x6     6 7 5 6 6 6 5 7 6 5 6 7
*    Block x7     6 7 5 5 7 6 7 6 5 6 6 6
*    Block x8     6 6 6 7 5 6 7 6 5 6 6 6
*    Block x9     7 5 6 5 5 8 6 6 6 5 6 7
*    Block x10    5 7 6 6 6 6 6 7 5 7 6 5
*    Block x11    4 7 7 7 5 6 5 6 7 7 6 5
*    Block x12    5 6 7 7 6 5 5 6 7 7 5 6
*    Block x13    6 6 6 6 5 7 6 6 6 6 6 6
*    Block x14    3 5 4 6 5 5 5 3 4 5 6 3 5 4 4 5
*    Block x15    6 6 6 6 5 7 5 7 6 7 6 5
     Block x16    9 9 9 9 9 9 9 9
*    Block x17    9 9 9 9 10 8 9 9
.
.
.

     N-Way        1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

They should be examined to ensure that each level is well represented in each block. The design is nearly balanced in most of the factors and blocks.

Collecting data is time consuming and expensive. Before collecting data, it is a good practice to convert your linear arrangement into a choice design and evaluate it in the context of a choice model. We start by creating some formats for the factor levels and the key to converting the linear arrangement into a choice design as follows:*

```
proc format;
   value price 1 = ' 999'      2 = '1249' 3 = '1499' 4 = '1749'  . = ' ';
   value scene 1 = 'Mountains' 2 = 'Lake'            3 = 'Beach' . = ' ';
   value lodge 1 = 'Cabin'     2 = 'Bed & Breakfast' 3 = 'Hotel' . = ' ';
   value side  1 = 'Side Trip' 2 = 'No'                          . = ' ';
   run;

data key;
   input Place $ 1-10 (Lodge Scene Price Side) ($);
   datalines;
Hawaii      x1  x6   x11  x16
Alaska      x2  x7   x12  .
Mexico      x3  x8   x13  x17
California  x4  x9   x14  .
Maine       x5  x10  x15  .
.           .   .    .    .
;
```

For analysis, the design has five attributes. `Place` is the alternative name. `Lodge`, `Scene`, `Price` and `Side` are created from the design using the indicated factors. See page 356 for more information about creating the design key. Notice that `Side` only applies to some of the alternatives and hence has missing values for the others. Processing the design and merging it with the data are similar to the examples on pages 356 and 371. One difference is now there are asymmetries in `Price`. For Hawaii's price, `x11`, we need to change 1, 2, and 3 to $1249, $1499, and $1749. For Alaska's price, `x12`, we need to change 1, 2, and 3 to $1249, $1499, and $1749. For Mexico's price, `x13`, we need to change 1, 2, and 3 to $999, $1249, and $1499. For California's price, `x14`, we need to change 1, 2, 3, and 4 to $999, $1249, $1499, and $1749. For Maine's price, `x11`, we need to change 1, 2, and 3 to $999, $1249, and $1499. We can simplify the problem by adding 1 to `x11` and `x12`, which are the factors that start at $1249 instead of $999. This lets us use a common format to set the price. See pages 499 and 884 for examples of handling more complicated asymmetries.

*See page 67 for an explanation of the linear arrangement of a choice design versus the arrangement of a choice design that is more suitable for analysis.

The following steps process the design:

```
data temp;
   set sasuser.AsymVacLinDesBlckd(rename=(block=Form));
   x11 + 1;
   x12 + 1;
   run;

%mktroll(design=temp, key=key, alt=place, out=sasuser.AsymVacChDes,
         options=nowarn, keep=form)

data sasuser.AsymVacChDes;
   set sasuser.AsymVacChDes;
   format scene scene. lodge lodge. side side. price price.;
   run;

proc print data=sasuser.AsymVacChDes(obs=12);
   by form set; id form set;
   run;
```

The first two choice sets are as follows:

```
                         Vacation Example with Asymmetry

Form    Set    Place         Lodge                Scene        Price    Side

  1      1     Hawaii        Bed & Breakfast      Lake         1499     No
               Alaska        Bed & Breakfast      Mountains    1249
               Mexico        Cabin                Lake          999     No
               California    Cabin                Lake         1249
               Maine         Hotel                Beach        1249


  1      2     Hawaii        Hotel                Lake         1749     Side Trip
               Alaska        Hotel                Lake         1499
               Mexico        Hotel                Beach        1249     No
               California    Cabin                Beach         999
               Maine         Cabin                Lake         1249
```

Notice that each has six alternatives, one of which is displaying in this format as all blank.

Testing the Design Before Data Collection

Collecting data is time consuming and expensive. It is always good practice to make sure that the design works with the most complicated model that we anticipate fitting. The following step evaluates the choice design:

```
title2 'Evaluate the Choice Design';

%choiceff(data=sasuser.AsymVacChDes,/* candidate set of choice sets         */
          init=sasuser.AsymVacChDes(keep=set), /* select these sets         */
          intiter=0,                /* evaluate without internal iterations */
                                    /* alternative-specific effects model   */
                                    /* zero=none - no ref levels for place  */
                                    /* order=data - do not sort levels      */
          model=class(place / zero=none order=data)
                                    /* zero=none - no ref levels any factor */
                                    /* order=formatted - sort levels        */
                                    /* use blank sep to build interact terms*/
                class(place * price place * scene place * lodge /
                      zero=none order=formatted separators='' ' ')
                                    /* no ref level for place               */
                                    /* ref level for side is 'No'           */
                                    /* use blank sep to build interact terms*/
                class(place * side / zero=' ' 'No' separators='' ' ') /
                lprefix=0           /* lpr=0 labels created from just levels*/
                cprefix=0,          /* cpr=0 names created from just levels */

          nsets=72,                 /* number of choice sets                */
          nalts=6,                  /* number of alternatives               */
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

We use the `%ChoicEff` macro to evaluate our choice design. You can both use this macro to search a candidate set for an efficient choice design, and you can use it to evaluate a design created by other means. The way you check a design like this is to first name it in the `data=` option. This is the candidate set that contains all of the choice sets that we will consider. In addition, the same design is named in the `init=` option. The full specification is `init=sasuser.AsymVacChDes(keep=set)`. Just the variable `Set` is kept. It is used to bring in just the indicated choice sets from the `data=` design, which in this case is all of them. The option `nsets=72` specifies the number of choice sets, and `nalts=6` specifies the number of alternatives. The option `beta=zero` specifies that we are assuming for design evaluation purpose that all of the betas or part-worth utilities are zero. You can evaluate the design for other parameter vectors by specifying a list of numbers after `beta=`. This changes the variances and standard errors. We also specify `intiter=0` which specifies zero internal iterations. We use zero internal iterations when we want to evaluate an initial design, but not attempt to improve it. The last option specifies the model.

The model specification contains everything that appears in the TRANSREG procedure's `model` statement for coding the design. Many of these options should be familiar from previous examples. The specification `class(place / zero=none order=data)` names the `place` variable as a classification variable and asks for coded variables for every nonmissing level (`zero=none`). The order of the levels on output matches the order that the levels are first encountered in the input data set. This specification creates the alternative effects or alternative-specific intercepts.

The next specification, `class(place * price place * scene place * lodge / zero=none order= formatted separators='' ' ')`, requests alternative-specific effects for all of the attributes except the side trip. The alternative-specific effects are requested by interacting the alternative-specific intercepts, in this case the destination, with the attributes. The `zero=none` option creates binary variables for all categories. In contrast, by default, a variable is not created for the last category—the parameter for the last category is a structural zero. The `zero=none` option is used when you want to see the structural zeros in the results. The `separators='' ' '` option (`separators=` quote quote space quote space quote, which provides two strings (one null and the other blank), allows you to specify two label component separators for the main effect and interaction terms, respectively. By specifying a blank for the second value, we request labels for the side trip effects like `'Mexico Side Trip'` instead of the default `'Mexico * Side Trip'`. This option is explained in more detail on page 449.

The last part of the model specification consists of `class(place * side / zero=' ' 'No' separators ='' ' ')` and creates the alternative-specific side trip effects with all levels for `place` and `'No'` as the reference level for the side trip attribute. See page 78 for more information about the `zero=` option. The last part of the model specification is followed by a slash and some options: `/ lprefix=0 cprefix=0)`. The `cprefix=0` option specifies that when names are created for the binary variables, zero characters of the original variable name should be used as a prefix. This means that the names are created only from the level values. The `lprefix=0` option specifies that when labels are created for the binary variables, zero characters of the original variable name should be used as a prefix. This means that the labels are created only from the level values.

The last part of the output is as follows:

```
                      Vacation Example with Asymmetry
                         Evaluate the Choice Design

                               Final Results

                         Design                  1
                         Choice Sets            72
                         Alternatives            6
                         Parameters             38
                         Maximum Parameters    360
                         D-Efficiency            0
                         D-Error                 .

                      Vacation Example with Asymmetry
                         Evaluate the Choice Design

                                                                     Standard
      n Variable Name             Label                   Variance DF   Error

      1 Hawaii                    Hawaii                   0.91061  1  0.95426
      2 Alaska                    Alaska                   0.70276  1  0.83831
      3 Mexico                    Mexico                   0.79649  1  0.89246
      4 California                California               0.89577  1  0.94645
      5 Maine                     Maine                    0.82172  1  0.90649
```

```
 6 Alaska_999                Alaska  999                    .        0   .
 7 Alaska_1249               Alaska 1249                  0.59635  1  0.77223
 8 Alaska_1499               Alaska 1499                  0.60551  1  0.77814
 9 Alaska_1749               Alaska 1749                    .        0   .
10 California_999            California  999              0.85492  1  0.92462
11 California_1249           California 1249              0.81130  1  0.90072
12 California_1499           California 1499              0.82552  1  0.90858
13 California_1749           California 1749                .        0   .
14 Hawaii_999                Hawaii  999                    .        0   .
15 Hawaii_1249               Hawaii 1249                  0.60792  1  0.77969
16 Hawaii_1499               Hawaii 1499                  0.59679  1  0.77252
17 Hawaii_1749               Hawaii 1749                    .        0   .
18 Maine_999                 Maine  999                   0.60676  1  0.77894
19 Maine_1249                Maine 1249                   0.61109  1  0.78172
20 Maine_1499                Maine 1499                     .        0   .
21 Maine_1749                Maine 1749                     .        0   .
22 Mexico_999                Mexico  999                  0.59178  1  0.76927
23 Mexico_1249               Mexico 1249                  0.60604  1  0.77849
24 Mexico_1499               Mexico 1499                    .        0   .
25 Mexico_1749               Mexico 1749                    .        0   .
26 AlaskaBeach               Alaska Beach                 0.63778  1  0.79861
27 AlaskaLake                Alaska Lake                  0.58330  1  0.76374
28 AlaskaMountains           Alaska Mountains               .        0   .
29 CaliforniaBeach           California Beach             0.59453  1  0.77106
30 CaliforniaLake            California Lake              0.67196  1  0.81973
31 CaliforniaMountains       California Mountains           .        0   .
32 HawaiiBeach               Hawaii Beach                 0.63923  1  0.79952
33 HawaiiLake                Hawaii Lake                  0.61115  1  0.78176
34 HawaiiMountains           Hawaii Mountains               .        0   .
35 MaineBeach                Maine Beach                  0.63688  1  0.79805
36 MaineLake                 Maine Lake                   0.58479  1  0.76471
37 MaineMountains            Maine Mountains                .        0   .
38 MexicoBeach               Mexico Beach                 0.59462  1  0.77111
39 MexicoLake                Mexico Lake                  0.59710  1  0.77272
40 MexicoMountains           Mexico Mountains               .        0   .
41 AlaskaBed___Breakfast     Alaska Bed & Breakfast       0.62130  1  0.78823
42 AlaskaCabin               Alaska Cabin                 0.61012  1  0.78110
43 AlaskaHotel               Alaska Hotel                   .        0   .
44 CaliforniaBed___Breakfast California Bed & Breakfast   0.62122  1  0.78817
45 CaliforniaCabin           California Cabin             0.60866  1  0.78016
46 CaliforniaHotel           California Hotel               .        0   .
47 HawaiiBed___Breakfast     Hawaii Bed & Breakfast       0.61876  1  0.78661
48 HawaiiCabin               Hawaii Cabin                 0.59145  1  0.76906
49 HawaiiHotel               Hawaii Hotel                   .        0   .
50 MaineBed___Breakfast      Maine Bed & Breakfast        0.61592  1  0.78480
51 MaineCabin                Maine Cabin                  0.60681  1  0.77898
52 MaineHotel                Maine Hotel                    .        0   .
53 MexicoBed___Breakfast     Mexico Bed & Breakfast       0.61050  1  0.78134
54 MexicoCabin               Mexico Cabin                 0.61670  1  0.78530
55 MexicoHotel               Mexico Hotel                   .        0   .
```

```
   56 AlaskaSide_Trip          Alaska Side Trip             .       0   .
   57 CaliforniaSide_Trip      California Side Trip         .       0   .
   58 HawaiiSide_Trip          Hawaii Side Trip         0.40413   1  0.63572
   59 MaineSide_Trip           Maine Side Trip              .       0   .
   60 MexicoSide_Trip          Mexico Side Trip         0.40622   1  0.63735
                                                                 ==
                                                                 38
```

It consists of a table with the name and label for each parameter along with its variance, *df*, and standard error. It needs to be carefully evaluated to see if the zeros and nonzeros are in all of the right places. We see one parameter for five of the six destinations, with the constant stay-at-home alternative in all cases excluded from the table. This is followed by four terms for the Alaska price effect. The Alaska at $999 parameter is zero since $999 does not apply to Alaska. The Alaska at $1749 parameter is the reference level and hence is zero. The other two Alaska price parameters are nonzero. Similarly, each of the alternative-specific price effects have two or three parameters (the number of applicable prices minus one). For the scenery and accommodations attributes, each alternative has two nonzero parameters and a reference level. There are two nonzero parameters for the side trips for the two applicable destinations. The pattern of zeros and nonzeros looks perfect. There are 38 parameters in the alternative-specific model.

You should also note that the variances and standard errors. They are all approximately the same order of magnitude. Sometimes you might see wildly varying parameters. This is usually a sign of a problematic design, perhaps one with too few choice sets for the number of parameters. This design looks good. The number of parameters in our model is 38 and the maximum number we could estimate with this design is 360, so too few choice sets should not be a problem. Note one difference between these results and the results that we see in the previous example on page 361. Here, our standard errors are not constant within an attribute, although they are similar. This is because none of our factors are orthogonal, although they are close.

Generating the Questionnaire

The following step produces the questionnaire:

```
%let m   = 6;                    /* m alts including constant  */
%let mm1 = %eval(&m - 1);        /* m - 1                      */
%let n   = 18;                   /* number of choice sets      */
%let blocks = 4;                 /* number of blocks           */


title;
options ls=80 ps=60 nonumber nodate;

data _null_;
   array dests[&mm1] $ 10 _temporary_ ('Hawaii' 'Alaska' 'Mexico'
                                        'California' 'Maine');
   array scenes[3]   $ 13 _temporary_
                     ('the Mountains' 'a Lake' 'the Beach');
```

```
array lodging[3]  $ 15 _temporary_
                  ('Cabin' 'Bed & Breakfast' 'Hotel');
array x[15];
array p[&mm1];
length price $ 6;
file print linesleft=ll;

set sasuser.AsymVacLinDesBlckd;
by block;

p1 = 1499 + (x[11] - 2) * 250;
p2 = 1499 + (x[12] - 2) * 250;
p3 = 1249 + (x[13] - 2) * 250;
p4 = 1374 + (x[14] - 2.5) * 250;
p5 = 1249 + (x[15] - 2) * 250;

if first.block then do;
   choice = 0;
   put _page_;
   put @50 'Form: ' block  ' Subject: ________' //;
   end;
choice + 1;

if ll < (19 + (x16 = 1) + (x17 = 1)) then put _page_;
put choice 2. ') Circle your choice of '
    'vacation destinations:' /;

do dest = 1 to &mm1;
   price = left(put(p[dest], dollar6.));
   put '    ' dest 1. ') ' dests[dest]
       +(-1) ', staying in a ' lodging[x[dest]]
       'near ' scenes[x[&mm1 + dest]] +(-1) ',' /
       +7 'with a package cost of ' price +(-1) @@;
   if dest = 3 and x16 = 1 then
      put ', and an optional visit' / +7
          'to archaeological sites for an additional $100' @@;
   else if dest = 1 and x17 = 1 then
      put ', and an optional helicopter' / +7
          'flight to an active volcano for an additional $200' @@;
   put '.' /;
   end;
put "    &m) Stay at home this year." /;
run;
```

The first two choice sets for the first subject are as follows:

```
                                                  Form: 1  Subject: ________


    1) Circle your choice of vacation destinations:

       1) Hawaii, staying in a Bed & Breakfast near a Lake,
          with a package cost of $1,499.

       2) Alaska, staying in a Bed & Breakfast near the Mountains,
          with a package cost of $1,249.

       3) Mexico, staying in a Cabin near a Lake,
          with a package cost of $999.

       4) California, staying in a Cabin near a Lake,
          with a package cost of $1,249.

       5) Maine, staying in a Hotel near the Beach,
          with a package cost of $1,249.

       6) Stay at home this year.

    2) Circle your choice of vacation destinations:

       1) Hawaii, staying in a Hotel near a Lake,
          with a package cost of $1,749.

       2) Alaska, staying in a Hotel near a Lake,
          with a package cost of $1,499.

       3) Mexico, staying in a Hotel near the Beach,
          with a package cost of $1,249, and an optional visit
          to archaeological sites for an additional $100.

       4) California, staying in a Cabin near the Beach,
          with a package cost of $999.

       5) Maine, staying in a Cabin near a Lake,
          with a package cost of $1,249.

       6) Stay at home this year.
```

In practice, data collection is typically be much more elaborate than this. It might involve art work or photographs, and the choice sets might be presented and the data might be collected through personal interview or over the Web. However the choice sets are presented and the data are collected, the essential elements remain the same. Subjects are shown a set of alternatives and are asked to make a choice, then they go on to the next set.

Generating Artificial Data

This section shows in a cursory way how to generate artificial data. Some researchers wisely like to use artificial data to test a design before spending lots of money on collecting data. See the section beginning on page 393 for a detailed discussion of generating artificial data.

The following step generates an artificial set of data:

```
data _null_;
   array dests[&mm1] _temporary_ (5 -1 4 3 2);
   array scenes[3]   _temporary_ (-1 0 1);
   array lodging[3]  _temporary_ (0 3 2);
   array u[&m];
   array x[15];
   do rep = 1 to 100;
      n = 0;
      do i = 1 to &blocks;
         k + 1;
         if mod(k,3) = 1 then put;
         put k 3. +1 i 1. +2 @@;
         do j = 1 to &n; n + 1;
            set sasuser.AsymVacLinDesBlckd point=n;
            do dest = 1 to &mm1;
               u[dest] = dests[dest] + lodging[x[dest]] +
                         scenes[x[&mm1 + dest]] -
                         x[2 * &mm1 + dest] +
                         2 * normal(17);
               end;
            u[1] = u[1] + (x16 = 1);
            u[3] = u[3] + (x17 = 1);
            u&m  = -3 + 3 * normal(17);
            m = max(of u1-u&m);
            if      abs(u1 - m) < 1e-4 then c = 1;
            else if abs(u2 - m) < 1e-4 then c = 2;
            else if abs(u3 - m) < 1e-4 then c = 3;
            else if abs(u4 - m) < 1e-4 then c = 4;
            else if abs(u5 - m) < 1e-4 then c = 5;
            else                            c = 6;
            put +(-1) c @@;
            end;
         end;
      end;
   stop;
   run;
```

Collecting data is time consuming and expensive. Generating some artificial data before the data are collected to test your code and make sure the analysis will run is a good idea. It helps avoid the "How am I going to analyze this?" question from occurring after the data have already been collected. This step generates data for 400 subjects, 100 per block.

The `dests`, `scenes`, and `lodging` arrays are initialized with part-worth utilities for each level. The utilities for each of the destinations are computed and stored in the array `u` in the statement `u[dest] = ...`, which includes an error term `2 * normal(17)`. The utilities for the side trips are added in separately with `u[1] = u[1] + (x16 = 1)` and `u[3] = u[3] + (x17 = 1)`. The utility for the stay-at-home alternative is `-3 + 3 * normal(17)`. The maximum utility is computed, `m = max(of u1-u&m)` and the alternative with the maximum utility is chosen. The `put` statement writes out the results to the log.

Reading, Processing, and Analyzing the Data

The results from the previous step are pasted into a DATA step and run to mimic reading real input data as follows:

```
title 'Vacation Example with Asymmetry';

data results;
   input Subj Form (choose1-choose&n) (1.) @@;
   datalines;
  1 1 413414111315351335   2 2 115311141441134121   3 3 331451344433513341
  4 4 113111143133311314   5 1 113413531545431313   6 2 145131111414331511
  7 3 313413113111313331   8 4 415143311133541321   9 1 133314111133431113
  .
  .
  .
;
```

The analysis proceeds in a fashion similar to the simpler vacation example on page 371. The following steps merge the data and display a subset of the results:

```
%mktmerge(design=sasuser.AsymVacChDes, data=results, out=res2, blocks=form,
          nsets=&n, nalts=&m, setvars=choose1-choose&n,
          stmts=%str(price = input(put(price, price.), 5.);
                     format scene scene. lodge lodge. side side.;))

proc print data=res2(obs=18);
   id form subj set; by form subj set;
   run;
```

The first three choice sets for the first subject are as follows:

```
                           Vacation Example with Asymmetry

     Form  Subj  Set  Place       Lodge             Scene      Price  Side       c

       1     1    1   Hawaii      Bed & Breakfast   Lake       1499   No         2
                      Alaska      Bed & Breakfast   Mountains  1249              2
                      Mexico      Cabin             Lake        999   No         2
                      California  Cabin             Lake       1249              1
                      Maine       Hotel             Beach      1249              2

       1     1    2   Hawaii      Hotel             Lake       1749   Side Trip  1
                      Alaska      Hotel             Lake       1499              2
                      Mexico      Hotel             Beach      1249   No         2
                      California  Cabin             Beach       999              2
                      Maine       Cabin             Lake       1249              2

       1     1    3   Hawaii      Hotel             Mountains  1749   Side Trip  2
                      Alaska      Hotel             Mountains  1749              2
                      Mexico      Bed & Breakfast   Beach      1249   Side Trip  1
                      California  Cabin             Lake       1249              2
                      Maine       Bed & Breakfast   Mountains   999              2
                                                                                 2
```

Indicator variables and labels are created using PROC TRANSREG like before as follows:

```
proc transreg design=5000 data=res2 nozeroconstant norestoremissing;
   model class(place / zero=none order=data)
         class(price scene lodge / zero=none order=formatted)
         class(place * side / zero=' ' 'No' separators='' ' ') /
         lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   id subj set form c;
   run;

proc print data=coded(obs=6) label;
   run;
```

The `design=5000` option specifies that no model is fit; the procedure is just being used to code a design in blocks of 5000 observations at a time. The `nozeroconstant` option specifies that if the coding creates a constant variable, it should not be zeroed. The `norestoremissing` option specifies that missing values should not be restored when the `out=` data set is created. The `model` statement names the variables to code and provides information about how they should be coded. The specification `class(place / ...)` specifies that the variable `Place` is a classification variable and requests a binary coding. The `zero=none` option creates binary variables for all categories. The `order=data` option sorts the levels into the order they are encountered in the data set. Similarly, the variables `Price`, `Scene`, and `Lodge` are classification variables. The specification `class(place * side / ...)` creates alternative-specific side trip effects. The option `zero=' ' 'No'` specifies that indicator variables should be created for all levels of `Place` except blank, and all levels of `Side` except `'No'`. The specification `zero=' '` is almost the same as `zero=none`. The `zero=' '` specification names a missing level as the reference level creating indicator variables for all nonmissing levels of the `class` variables, just like `zero=none`. The difference is `zero=none` applies to all of the variables named in the `class` specification. When you want `zero=none` to apply to only some variables, then you must use `zero=' '`, as in `zero=' ' 'No'` instead. In this case, `zero=none` applies to the first variable and `zero='No'` applies to the second. With `zero=' '`, PROC TRANSREG displays the following warning, which can be safely ignored:

```
   WARNING: Reference level ZERO='' was not found for variable Place.
```

See page 78 for more information about the `zero=` option.

The `separators='' ' '` option (`separators=` quote quote space quote space quote) allows you to specify two label component separators for the main effect and interaction terms, respectively. By specifying a blank for the second value, we request labels for the side trip effects like `'Mexico Side Trip'` instead of the default `'Mexico * Side Trip'`. This option is explained in more detail on page 449.

The `lprefix=0` option specifies that when labels are created for the binary variables, zero characters of the original variable name should be used as a prefix. This means that the labels are created only from the level values. An `output` statement names the output data set and drops variables that are not needed. Finally, the `id` statement names the additional variables that we want copied from the input to the output data set. The results are as follows:

```
                             Vacation Example with Asymmetry

             Obs Hawaii Alaska Mexico California Maine  999 1249 1499 1749 Beach Lake

               1    1      0      0        0        0     0    0    1    0    0     1
               2    0      1      0        0        0     0    1    0    0    0     0
               3    0      0      1        0        0     1    0    0    0    0     1
               4    0      0      0        1        0     0    1    0    0    0     1
               5    0      0      0        0        1     0    1    0    0    1     0
               6    0      0      0        0        0     0    0    0    0    0     0

                                                Alaska            Hawaii Maine Mexico
                               Bed &             Side  California  Side   Side  Side
             Obs Mountains Breakfast Cabin Hotel  Trip   Side Trip  Trip   Trip  Trip

               1     0          1        0     0     0        0        0     0     0
               2     1          1        0     0     0        0        0     0     0
               3     0          0        1     0     0        0        0     0     0
               4     0          0        1     0     0        0        0     0     0
               5     0          0        0     1     0        0        0     0     0
               6     0          0        0     0     0        0        0     0     0
```

```
Obs Place       Price  Scene      Lodge            Side  Subj  Set  Form  c

  1 Hawaii      1499   Lake       Bed & Breakfast   No     1     1     1   2
  2 Alaska      1249   Mountains  Bed & Breakfast          1     1     1   2
  3 Mexico       999   Lake       Cabin             No     1     1     1   2
  4 California  1249   Lake       Cabin                    1     1     1   1
  5 Maine       1249   Beach      Hotel                    1     1     1   2
  6                                                        1     1     1   2
```

The PROC PHREG specification is the same as we have used before. Recall that we used `%phchoice(on)` on page 287 to customize the output from PROC PHREG.) The following step analyzes the data:

```
proc phreg data=coded brief;
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;
```

The results are as follows:

```
                        Vacation Example with Asymmetry

                              The PHREG Procedure

                               Model Information

                     Data Set                 WORK.CODED
                     Dependent Variable       c
                     Censoring Variable       c
                     Censoring Value(s)       2
                     Ties Handling            BRESLOW

                   Number of Observations Read       43200
                   Number of Observations Used       43200

         Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                   Number of       Number of          Chosen          Not
        Pattern      Choices    Alternatives    Alternatives       Chosen

              1         7200               6               1            5

                              Convergence Status

                    Convergence criterion (GCONV=1E-8) satisfied.
```

```
                      Model Fit Statistics

                                 Without           With
                  Criterion     Covariates     Covariates

                  -2 LOG L       25801.336      12603.247
                  AIC            25801.336      12631.247
                  SBC            25801.336      12727.593

                  Testing Global Null Hypothesis: BETA=0

          Test                 Chi-Square       DF     Pr > ChiSq

          Likelihood Ratio     13198.0891       14         <.0001
          Score                12223.0125       14         <.0001
          Wald                  5081.3295       14         <.0001

                   Multinomial Logit Parameter Estimates

                              Parameter     Standard
                        DF     Estimate        Error   Chi-Square   Pr > ChiSq

Hawaii                   1      3.61141      0.22224     264.0701       <.0001
Alaska                   1     -0.94997      0.26364      12.9836       0.0003
Mexico                   1      2.26877      0.22776      99.2247       <.0001
California               1      1.54548      0.22760      46.1102       <.0001
Maine                    1      0.74153      0.23210      10.2074       0.0014
 999                     1      2.10214      0.07298     829.7619       <.0001
1249                     1      1.44298      0.06078     563.6949       <.0001
1499                     1      0.72311      0.05936     148.4188       <.0001
1749                     0            0            .            .            .
Beach                    1      1.42021      0.04635     938.8384       <.0001
Lake                     1      0.72019      0.04472     259.3676       <.0001
Mountains                0            0            .            .            .
Bed & Breakfast          1      0.65045      0.04079     254.3369       <.0001
Cabin                    1     -1.42317      0.04809     875.8795       <.0001
Hotel                    0            0            .            .            .
Alaska Side Trip         0            0            .            .            .
California Side Trip     0            0            .            .            .
Hawaii Side Trip         1      0.71850      0.05753     155.9801       <.0001
Maine Side Trip          0            0            .            .            .
Mexico Side Trip         1      0.65550      0.06293     108.4863       <.0001
```

You would not expect the part-worth utilities to match those that are used to generate the data, but you would expect a similar ordering within each attribute, and in fact that does occur. These data can also be analyzed with quantitative price effects and destination by attribute interactions, as in the previous vacation example.

Aggregating the Data

This data set is rather large with 43,200 observations. You can make the analysis run faster and with less memory by aggregating. Instead of stratifying on each choice set and subject combination, you can stratify just on choice set and specify the number of times each alternative is chosen and the number of times it is not chosen. First, use PROC SUMMARY to count the number of times each observation occurs. Specify all the analysis variables, and in this example, also specify `Form`. The variable `Form` is added to the list because `Set` designates choice set within form. It is the `Form` and `Set` combinations that identify the choice sets. (In the previous PROC PHREG step, since the `Subj * Set` combinations uniquely identified each stratum, `Form` is not needed.) PROC SUMMARY stores the number of times each unique observation appears in the variable `_freq_`. PROC PHREG is then run with a `freq` statement. Now, instead of analyzing a data set with 43,200 observations and 7200 strata, we analyze a data set with at most $2 \times 6 \times 72 = 864$ observations and 72 strata. For each of the 6 alternatives and 72 choice sets, there are typically 2 observations in the aggregate data set: one that contains the number of times it is chosen and one that contains the number of times it is not chosen. When one of those counts is zero, there is one observation. In this case, the aggregate data set has 724 observations. The following steps aggregate and perform the analysis:

```
proc summary data=coded nway;
   class form set c &_trgind;
   output out=agg(drop=_type_);
   run;


proc phreg data=agg;
   model c*c(2) = &_trgind / ties=breslow;
   freq _freq_;
   strata form set;
   run;
```

PROC SUMMARY ran in three seconds, and PROC PHREG ran in less than one second. The results are as follows:

```
                    Vacation Example with Asymmetry

                          The PHREG Procedure

                           Model Information

                  Data Set                    WORK.AGG
                  Dependent Variable          c
                  Censoring Variable          c
                  Censoring Value(s)          2
                  Frequency Variable          _FREQ_
                  Ties Handling               BRESLOW

                Number of Observations Read         724
                Number of Observations Used         724
                Sum of Frequencies Read           43200
                Sum of Frequencies Used           43200
```

```
              Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                                          Number of           Chosen         Not
Stratum    Form        Set             Alternatives     Alternatives      Chosen

      1    1           1                        600              100         500
      2    1           2                        600              100         500
      3    1           3                        600              100         500
      4    1           4                        600              100         500
      .
      .
      .
     71    4           71                       600              100         500
     72    4           72                       600              100         500
---------------------------------------------------------------------------------
  Total                                       43200             7200       36000
```

The parameter estimates and Chi-Square statistics (not shown) are the same as before. The summary table shows the results of the aggregation, 100 out of 600 alternatives are chosen in each stratum. The log likelihood statistics are different, but that does not matter since the Chi-Square statistics are the same. Page 466 provides more information about this.

Brand Choice Example with Aggregate Data

In this next example, subjects are presented with brands of a product at different prices. There are four brands and a constant alternative, eight choice sets, and 100 subjects. This example shows how to handle data that come to you already aggregated. It also illustrates comparing the fits of two competing models, the mother logit model, cross-effects, IIA, and techniques for handling large data sets. The choice sets, with the price of each alternative and the number of times it is chosen, are shown next.

Set	Brand 1		Brand 2		Brand 3		Brand 4		Other	
1	$3.99	4	$5.99	29	$3.99	16	$5.99	42	$4.99	9
2	$5.99	12	$5.99	19	$5.99	22	$5.99	33	$4.99	14
3	$5.99	34	$5.99	26	$3.99	8	$3.99	27	$4.99	5
4	$5.99	13	$3.99	37	$5.99	15	$3.99	27	$4.99	8
5	$5.99	49	$3.99	1	$3.99	9	$5.99	37	$4.99	4
6	$3.99	31	$5.99	12	$5.99	6	$3.99	18	$4.99	33
7	$3.99	37	$3.99	10	$5.99	5	$5.99	35	$4.99	13
8	$3.99	16	$3.99	14	$3.99	5	$3.99	51	$4.99	14

The first choice set consists of Brand 1 at $3.99, Brand 2 at $5.99, Brand 3 at $3.99, Brand 4 at $5.99, and Other at $4.99. From this choice set, Brand 1 is chosen 4 times, Brand 2 is chosen 29 times, Brand 3 is chosen 16 times, Brand 4 is chosen 42 times, and Other is chosen 9 times.

Processing the Data

As in the previous examples, we process the data to create a data set with one stratum for each choice set within each subject and m alternatives per stratum. This example has 100 people times 5 alternatives times 8 choice sets equals 4000 observations. The first five observations are for the first subject and the first choice set, the next five observations are for the second subject and the first choice set, ..., the next five observations are for the one-hundredth subject and the first choice set, the next five observations are for the first subject and the second choice set, and so on. Subject 1 in the first choice set is almost certainly not the same as subject 1 in subsequent choice sets since we are given aggregate data. However, that is not important. What is important is that we have a subject and choice set variable whose unique combinations identify each choice set within each subject. In previous examples, we specified `strata Subj Set` with PROC PHREG, and our data are sorted by choice set within subject. We can still use the same specification even though our data are now sorted by subject within choice set. The following step reads and prepares the data:

```
%let m = 5;  /* Number of Brands in Each Choice Set */
             /* (including Other)                   */

title 'Brand Choice Example, Multinomial Logit Model';

proc format;
   value brand 1 = 'Brand 1' 2 = 'Brand 2' 3 = 'Brand 3'
               4 = 'Brand 4' 5 = 'Other';
   run;
```

```
data price;
   array p[&m] p1-p&m; /* Prices for the Brands */
   array f[&m] f1-f&m; /* Frequency of Choice   */

   input p1-p&m f1-f&m;
   keep subj set brand price c p1-p&m;

   * Store choice set and subject number to stratify;
   Set = _n_; Subj = 0;

   do i = 1 to &m;           /* Loop over the &m frequencies    */
      do ci = 1 to f[i];     /* Loop frequency of choice times  */
         subj + 1;           /* Subject within choice set       */
         do Brand = 1 to &m; /* Alternatives within choice set  */

            Price = p[brand];

            * Output first choice: c=1, unchosen: c=2;
            c = 2 - (i eq brand); output;
            end;
         end;
      end;

format brand brand.;

datalines;
3.99 5.99 3.99 5.99 4.99   4 29 16 42  9
5.99 5.99 5.99 5.99 4.99  12 19 22 33 14
5.99 5.99 3.99 3.99 4.99  34 26  8 27  5
5.99 3.99 5.99 3.99 4.99  13 37 15 27  8
5.99 3.99 3.99 5.99 4.99  49  1  9 37  4
3.99 5.99 5.99 3.99 4.99  31 12  6 18 33
3.99 3.99 5.99 5.99 4.99  37 10  5 35 13
3.99 3.99 3.99 3.99 4.99  16 14  5 51 14
;


proc print data=price(obs=15);
   var subj set c price brand;
   run;
```

The inner loop `do Brand = 1 to &m` creates all of the observations for the m alternatives within a person/choice set combination. Within a choice set (row of input data), the outer two loops, `do i = 1 to &m` and `do ci = 1 to f[i]` execute the code inside 100 times, the variable `Subj` goes from 1 to 100. In the first choice set, they first create the data for the four subjects that chose Brand 1, then the data for the 29 subjects that chose Brand 2, and so on. The first 15 observations of the data set are as follows:

```
                    Brand Choice Example, Multinomial Logit Model

                      Obs    Subj    Set    c    Price     Brand

                        1      1      1     1     3.99    Brand 1
                        2      1      1     2     5.99    Brand 2
                        3      1      1     2     3.99    Brand 3
                        4      1      1     2     5.99    Brand 4
                        5      1      1     2     4.99    Other
                        6      2      1     1     3.99    Brand 1
                        7      2      1     2     5.99    Brand 2
                        8      2      1     2     3.99    Brand 3
                        9      2      1     2     5.99    Brand 4
                       10      2      1     2     4.99    Other
                       11      3      1     1     3.99    Brand 1
                       12      3      1     2     5.99    Brand 2
                       13      3      1     2     3.99    Brand 3
                       14      3      1     2     5.99    Brand 4
                       15      3      1     2     4.99    Other
```

Note that the data set also contains the variables `p1-p5` which contain the prices of each of the alternatives. These variables, which are used in constructing the cross-effects, are discussed in more detail on page 452. The following step displays the first five rows:

```
proc print data=price(obs=5); run;
```

The results are as follows:

```
                    Brand Choice Example, Multinomial Logit Model

      Obs    p1      p2      p3      p4      p5     Set    Subj     Brand      Price    c

        1   3.99    5.99    3.99    5.99    4.99     1       1     Brand 1     3.99    1
        2   3.99    5.99    3.99    5.99    4.99     1       1     Brand 2     5.99    2
        3   3.99    5.99    3.99    5.99    4.99     1       1     Brand 3     3.99    2
        4   3.99    5.99    3.99    5.99    4.99     1       1     Brand 4     5.99    2
        5   3.99    5.99    3.99    5.99    4.99     1       1     Other       4.99    2
```

Simple Price Effects

The data are coded using PROC TRANSREG as follows:

```
proc transreg design data=price nozeroconstant norestoremissing;
   model class(brand / zero=none) identity(price) / lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   label price = 'Price';
   id subj set c;
   run;
```

The `design` option specifies that no model is fit; the procedure is just being used to code a design. The `nozeroconstant` option specifies that if the coding creates a constant variable, it should not be zeroed. The `norestoremissing` option specifies that missing values should not be restored when the `out=` data set is created. The `model` statement names the variables to code and provides information about how they should be coded. The specification `class(brand / zero=none)` specifies that the variable `Brand` is a classification variable and requests a binary coding. The `zero=none` option creates binary variables for all categories. The specification `identity(price)` specifies that the variable `Price` is quantitative and hence should directly enter the model without coding. The `lprefix=0` option specifies that when labels are created for the binary variables, zero characters of the original variable name should be used as a prefix. This means that the labels are created only from the level values. An `output` statement names the output data set and drops variables that are not needed. Finally, the `id` statement names the additional variables that we want copied from the input to the output data set. The following step performs the analysis:

```
proc phreg data=coded brief;
   title2 'Discrete Choice with Common Price Effect';
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;
```

Recall that we used `%phchoice(on)` on page 287 to customize the output from PROC PHREG. The results are as follows:

```
                Brand Choice Example, Multinomial Logit Model
                   Discrete Choice with Common Price Effect

                             The PHREG Procedure

                              Model Information

                     Data Set                  WORK.CODED
                     Dependent Variable        c
                     Censoring Variable        c
                     Censoring Value(s)        2
                     Ties Handling             BRESLOW

                   Number of Observations Read        4000
                   Number of Observations Used        4000
```

```
        Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                 Number of       Number of          Chosen         Not
        Pattern    Choices    Alternatives    Alternatives      Chosen

              1        800               5               1           4

                               Convergence Status

                 Convergence criterion (GCONV=1E-8) satisfied.

                              Model Fit Statistics

                                  Without           With
                   Criterion   Covariates     Covariates

                   -2 LOG L      2575.101       2425.214
                   AIC           2575.101       2435.214
                   SBC           2575.101       2458.637

                   Testing Global Null Hypothesis: BETA=0

           Test                 Chi-Square       DF     Pr > ChiSq

           Likelihood Ratio       149.8868        5         <.0001
           Score                  153.2328        5         <.0001
           Wald                   142.9002        5         <.0001

                    Multinomial Logit Parameter Estimates

                       Parameter      Standard
               DF       Estimate         Error    Chi-Square    Pr > ChiSq

     Brand 1    1        0.66727       0.12305       29.4065        <.0001
     Brand 2    1        0.38503       0.12962        8.8235        0.0030
     Brand 3    1       -0.15955       0.14725        1.1740        0.2786
     Brand 4    1        0.98964       0.11720       71.2993        <.0001
     Other      0              0             .             .             .
     Price      1        0.14966       0.04406       11.5379        0.0007
```

Alternative-Specific Price Effects

In the following step, the data are coded for fitting a multinomial logit model with brand by price effects:

```
proc transreg design data=price nozeroconstant norestoremissing;
   model class(brand / zero=none separators='' ' ') |
         identity(price) / lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   label price = 'Price';
   id subj set c;
   run;
```

The PROC TRANSREG `model` statement has a vertical bar, "`|`", between the `class` specification and the `identity` specification. Since the `zero=none` option is specified with `class`, the vertical bar creates two sets of variables: five indicator variables for the brand effects and five more variables for the brand by price interactions. The `separators=` option allows you to specify two label component separators as quoted strings. The specification `separators='' ' '` (`separators=` quote quote space quote space quote) specifies a null string (quote quote) and a blank (quote space quote). The `separators='' ' '` option in the `class` specification specifies the separators that are used to construct the labels for the main effect and interaction terms, respectively. By default, the alternative-specific price effects—the brand by price interactions—have labels like `'Brand 1 * Price'` since the default second value for `separators=` is `' * '` (a quoted space asterisk space). Specifying `' '` (a quoted space) as the second value creates labels of the form `'Brand 1 Price'`. Since `lprefix=0`, the main-effects separator, which is the first `separators=` value, `''` (quote quote), is ignored. Zero name or input variable label characters are used to construct the label. The label is simply the formatted value of the `class` variable. The following steps display the first two coded choice sets and perform the analysis:

```
proc print data=coded(obs=10) label;
   title2 'Discrete Choice with Brand by Price Effects';
   var subj set c brand price &_trgind;
   run;

proc phreg data=coded brief;
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;

title2;
```

The results are as follows:

```
                      Brand Choice Example, Multinomial Logit Model
                       Discrete Choice with Brand by Price Effects

                                                        Brand    Brand    Brand    Brand
    Obs    Subj    Set    c     Brand      Price        1        2        3        4

      1      1      1     1    Brand 1     3.99         1        0        0        0
      2      1      1     2    Brand 2     5.99         0        1        0        0
      3      1      1     2    Brand 3     3.99         0        0        1        0
      4      1      1     2    Brand 4     5.99         0        0        0        1
      5      1      1     2    Other       4.99         0        0        0        0
      6      2      1     1    Brand 1     3.99         1        0        0        0
      7      2      1     2    Brand 2     5.99         0        1        0        0
      8      2      1     2    Brand 3     3.99         0        0        1        0
      9      2      1     2    Brand 4     5.99         0        0        0        1
     10      2      1     2    Other       4.99         0        0        0        0

                      Brand 1    Brand 2    Brand 3    Brand 4    Other
    Obs    Other       Price      Price      Price      Price     Price

      1      0         3.99       0.00       0.00       0.00      0.00
      2      0         0.00       5.99       0.00       0.00      0.00
      3      0         0.00       0.00       3.99       0.00      0.00
      4      0         0.00       0.00       0.00       5.99      0.00
      5      1         0.00       0.00       0.00       0.00      4.99
      6      0         3.99       0.00       0.00       0.00      0.00
      7      0         0.00       5.99       0.00       0.00      0.00
      8      0         0.00       0.00       3.99       0.00      0.00
      9      0         0.00       0.00       0.00       5.99      0.00
     10      1         0.00       0.00       0.00       0.00      4.99

                      Brand Choice Example, Multinomial Logit Model
                       Discrete Choice with Brand by Price Effects

                                  The PHREG Procedure

                                   Model Information

                           Data Set                 WORK.CODED
                           Dependent Variable       c
                           Censoring Variable       c
                           Censoring Value(s)       2
                           Ties Handling            BRESLOW

                         Number of Observations Read        4000
                         Number of Observations Used        4000
```

```
        Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                   Number of       Number of          Chosen         Not
        Pattern      Choices    Alternatives    Alternatives      Chosen

              1          800               5               1           4

                              Convergence Status

                   Convergence criterion (GCONV=1E-8) satisfied.

                            Model Fit Statistics

                                     Without          With
                      Criterion   Covariates    Covariates

                      -2 LOG L      2575.101      2424.812
                      AIC           2575.101      2440.812
                      SBC           2575.101      2478.288

                      Testing Global Null Hypothesis: BETA=0

            Test                 Chi-Square       DF     Pr > ChiSq

            Likelihood Ratio       150.2891        8         <.0001
            Score                  154.2563        8         <.0001
            Wald                   143.1425        8         <.0001

                       Multinomial Logit Parameter Estimates

                            Parameter     Standard
                      DF     Estimate        Error    Chi-Square    Pr > ChiSq

      Brand 1          1     -0.00972      0.43555        0.0005        0.9822
      Brand 2          1     -0.62230      0.48866        1.6217        0.2028
      Brand 3          1     -0.81250      0.60318        1.8145        0.1780
      Brand 4          1      0.31778      0.39549        0.6456        0.4217
      Other            0            0            .             .             .
      Brand 1 Price    1      0.13587      0.08259        2.7063        0.1000
      Brand 2 Price    1      0.20074      0.09210        4.7512        0.0293
      Brand 3 Price    1      0.13126      0.11487        1.3057        0.2532
      Brand 4 Price    1      0.13478      0.07504        3.2255        0.0725
      Other Price      0            0            .             .             .
```

The likelihood for this model is essentially the same as for the simpler, common-price-slope model fit previously, $-2\log(\mathcal{L}_C) = 2425.214$ compared to 2424.812. You can test the null hypothesis that the two models are not significantly different by comparing their likelihoods. The difference between two $-2\log(\mathcal{L}_C)$'s (the number reported under 'With Covariates' in the output) has a chi-square distribution. We can get the *df* for the test by subtracting the two *df* for the two likelihoods. The difference $2425.214 - 2424.812 = 0.402$ is distributed χ^2 with $8 - 5 = 3$ *df* and is not statistically significant.

Mother Logit Model

This next step fits the so-called "mother logit" model. This step creates the full design matrix, including the brand, price, and cross-effects. A cross-effect represents the effect of one alternative on the utility of another alternative. First, the input data set for the first choice set is displayed by the following step:

```
proc print data=price(obs=5) label;
   run;
```

The results are as follows:

```
                  Brand Choice Example, Multinomial Logit Model

     Obs    p1      p2      p3      p4      p5     Set    Subj     Brand      Price    c

       1   3.99    5.99    3.99    5.99    4.99     1       1     Brand 1     3.99    1
       2   3.99    5.99    3.99    5.99    4.99     1       1     Brand 2     5.99    2
       3   3.99    5.99    3.99    5.99    4.99     1       1     Brand 3     3.99    2
       4   3.99    5.99    3.99    5.99    4.99     1       1     Brand 4     5.99    2
       5   3.99    5.99    3.99    5.99    4.99     1       1     Other       4.99    2
```

The input consists of `Set`, `Subj`, `Brand`, `Price`, and a choice time variable `c`. In addition, it contains five variables `p1` through `p5`. The first observation of the `Price` variable shows us that the first alternative costs \$3.99; `p1` contains the cost of alternative 1, \$3.99, which is the same for all alternatives. It does not matter which alternative you are looking at, `p1` shows that alternative 1 costs \$3.99. Similarly, the second observation of the `Price` variable shows us that the second alternative costs \$5.99; `p2` contains the cost of alternative 2, \$5.99, which is the same for all alternatives. There is one price variable, `p1` through `p5`, for each of the five alternatives.

In all of the previous examples, we use models that are coded so that the utility of an alternative only depends on the attributes of that alternative. For example, the utility of Brand 1 only depends on the Brand 1 name and its price. In contrast, `p1-p5` contain information about each of the *other* alternatives' attributes. We construct cross-effects using the interaction of `p1-p5` and the `Brand` variable. In a model with cross-effects, the utility for an alternative depends on both that alternative's attributes *and* the other alternatives' attributes. The IIA (independence from irrelevant alternatives) property states that utility only depends on an alternative's own attributes. Cross-effects add other alternative's attributes to the model, so they can be used to test for violations of IIA. (See pages 459, 468, 674, and 679 for other discussions of IIA.) The PROC TRANSREG step for coding the cross-effects model is as follows:

```
proc transreg design data=price nozeroconstant norestoremissing;
   model class(brand / zero=none separators='' ' ') | identity(price)
         identity(p1-p&m) *
            class(brand / zero=none lprefix=0 separators='' ' on ') /
         lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   label price = 'Price'
         p1 = 'Brand 1' p2 = 'Brand 2' p3 = 'Brand 3'
         p4 = 'Brand 4' p5 = 'Other';
   id subj set c;
   run;
```

The `class(brand / ...) | identity(price)` specification in the `model` statement is the same as the previous analysis. The additional terms, `identity(p1-p&m) * class(brand / ...)` create the cross-effects. The second value of the `separators=` option, `' on'` is used to create labels like `'Brand 1 on Brand 2'` instead of the default `'Brand 1 * Brand 2'`. It is important to note that you must specify the cross-effect by specifying `identity` with the price factors, followed by the asterisk, followed by `class` and the brand effect, *in that order*. The order of the specification determines the order in which brand names are added to the labels. Do not specify the brand variable first; doing so creates incorrect labels.

With m alternatives, there are $m \times m$ cross-effects, but as we will see, many of them are zero. The first coded choice set is displayed with the following PROC PRINT steps:

```
title2 'Discrete Choice with Cross-Effects, Mother Logit';
proc format; value zer 0 = '  0' 1 = '  1'; run;
proc print data=coded(obs=5) label; var subj set c brand price; run;
proc print data=coded(obs=5) label; var Brand:;
   format brand: zer5.2 brand brand.; run;
proc print data=coded(obs=5) label; var p1B:; format p: zer5.2; id brand; run;
proc print data=coded(obs=5) label; var p2B:; format p: zer5.2; id brand; run;
proc print data=coded(obs=5) label; var p3B:; format p: zer5.2; id brand; run;
proc print data=coded(obs=5) label; var p4B:; format p: zer5.2; id brand; run;
proc print data=coded(obs=5) label; var p5B:; format p: zer5.2; id brand; run;
```

The coded data set contains the strata variable `Subj` and `Set`, choice time variable `c`, and `Brand` and `Price`. `Brand` and `Price` are used to create the coded independent variables but they are not used in the analysis with PROC PHREG.

The results are as follows:

```
                    Brand Choice Example, Multinomial Logit Model
                   Discrete Choice with Cross-Effects, Mother Logit

                      Obs    Subj    Set    c     Brand      Price

                        1      1      1     1    Brand 1     3.99
                        2      1      1     2    Brand 2     5.99
                        3      1      1     2    Brand 3     3.99
                        4      1      1     2    Brand 4     5.99
                        5      1      1     2    Other       4.99
```

The effects `'Brand 1'` through `'Other'` in the next output are the binary brand effect variables. They indicate the brand for each alternative. The effects `'Brand 1 Price'` through `'Other Price'` are alternative-specific price effects. They indicate the price for each alternative. All ten of these variables are independent variables in the analysis, and their names are part of the `&_trgind` macro variable list, as are all of the cross-effects that are described next. The results are as follows:

```
                    Brand Choice Example, Multinomial Logit Model
                   Discrete Choice with Cross-Effects, Mother Logit

        Brand Brand Brand Brand       Brand 1 Brand 2 Brand 3 Brand 4 Other
   Obs    1     2     3     4   Other  Price   Price   Price   Price  Price  Brand

     1    1     0     0     0     0     3.99     0       0       0      0    Brand 1
     2    0     1     0     0     0      0      5.99     0       0      0    Brand 2
     3    0     0     1     0     0      0       0      3.99     0      0    Brand 3
     4    0     0     0     1     0      0       0       0      5.99    0    Brand 4
     5    0     0     0     0     1      0       0       0       0     4.99 Other
```

The effects `'Brand 1 on Brand 1'` through `'Brand 1 on Other'` in the next output are the first five cross-effects.

The results are as follows:

```
                 Brand Choice Example, Multinomial Logit Model
                Discrete Choice with Cross-Effects, Mother Logit

                  Brand      Brand      Brand      Brand
                  1 on       1 on       1 on       1 on      Brand 1
        Brand    Brand 1    Brand 2    Brand 3    Brand 4    on Other

       Brand 1     3.99        0          0          0          0
       Brand 2     0         3.99         0          0          0
       Brand 3     0           0        3.99         0          0
       Brand 4     0           0          0        3.99         0
       Other       0           0          0          0        3.99
```

They represent the effect of Brand 1 at its price on the utility of each alternative. The label `'Brand` *n* `on Brand` *m*`'` is read as 'the effect of Brand *n* at its price on the utility of Brand *m*.' For the first choice set, these first five cross-effects consist entirely of zeros and \$3.99's, where \$3.99 is the price of Brand 1 in this choice set. The nonzero value is constant across all of the alternatives in each choice set since Brand 1 has only one price in each choice set. Notice the `'Brand 1 on Brand 1'` term, which is the effect of Brand 1 at its price on the utility of Brand 1. Also notice the `'Brand 1 Price'` effect, which is shown in the previous output. The description "the effect of Brand 1 at its price on the utility of Brand 1" is just a convoluted way of describing the Brand 1 price effect. The `'Brand 1 on Brand 1'` cross-effect is the same as the Brand 1 price effect, hence when we do the analysis, we see that the coefficient for the `'Brand 1 on Brand 1'` cross-effect is zero.

The effects `'Brand 2 on Brand 1'` through `'Brand 2 on Other'` in the next output are the next five cross-effects. The results are as follows:

```
                 Brand Choice Example, Multinomial Logit Model
                Discrete Choice with Cross-Effects, Mother Logit

                  Brand      Brand      Brand      Brand
                  2 on       2 on       2 on       2 on      Brand 2
        Brand    Brand 1    Brand 2    Brand 3    Brand 4    on Other

       Brand 1     5.99        0          0          0          0
       Brand 2     0         5.99         0          0          0
       Brand 3     0           0        5.99         0          0
       Brand 4     0           0          0        5.99         0
       Other       0           0          0          0        5.99
```

They represent the effect of Brand 2 at its price on the utility of each alternative. For the first choice set, these five cross-effects consist entirely of zeros and \$5.99's, where \$5.99 is the price of Brand 2 in this choice set. The nonzero value is constant across all of the alternatives in each choice set since Brand 2 has only one price in each choice set. Notice the `'Brand 2 on Brand 2'` term, which is the effect of Brand 2 at its price on the utility of Brand 2. The description "the effect of Brand 2 at its price on the utility of Brand 2" is just a convoluted way of describing the Brand 2 price effect. The `'Brand 2 on Brand 2'` cross-effect is the same as the Brand 2 price effect, hence when we do the analysis, we see that the coefficient for the `'Brand 2 on Brand 2'` cross-effect is zero.

The effects `'Brand 3 on Brand 1'` through `'Brand 3 on Other'` in the next output are the next five cross-effects. The results are as follows:

```
                 Brand Choice Example, Multinomial Logit Model
                Discrete Choice with Cross-Effects, Mother Logit

                Brand      Brand      Brand      Brand
                3 on       3 on       3 on       3 on      Brand 3
     Brand     Brand 1    Brand 2    Brand 3    Brand 4    on Other

    Brand 1      3.99        0          0          0          0
    Brand 2     0          3.99         0          0          0
    Brand 3     0           0          3.99        0          0
    Brand 4     0           0           0         3.99        0
    Other       0           0           0          0         3.99
```

They represent the effect of Brand 3 at its price on the utility of each alternative. For the first choice set, these five cross-effects consist entirely of zeros and $3.99's, where $3.99 is the price of Brand 3 in this choice set. Notice that the `'Brand 3 on Brand 3'` term is the same as the Brand 3 price effect, hence when we do the analysis, see that the coefficient for the `'Brand 3 on Brand 3'` cross-effect is zero.

The remaining cross-effects follow the same pattern that was described for the previous cross-effects and are as follows:

```
                 Brand Choice Example, Multinomial Logit Model
                Discrete Choice with Cross-Effects, Mother Logit

                Brand      Brand      Brand      Brand
                4 on       4 on       4 on       4 on      Brand 4
     Brand     Brand 1    Brand 2    Brand 3    Brand 4    on Other

    Brand 1      5.99        0          0          0          0
    Brand 2     0          5.99         0          0          0
    Brand 3     0           0          5.99        0          0
    Brand 4     0           0           0         5.99        0
    Other       0           0           0          0         5.99
```

```
                    Brand Choice Example, Multinomial Logit Model
                   Discrete Choice with Cross-Effects, Mother Logit

                    Other on    Other on    Other on    Other on    Other on
          Brand     Brand 1     Brand 2     Brand 3     Brand 4      Other

         Brand 1      4.99         0           0           0           0
         Brand 2       0          4.99         0           0           0
         Brand 3       0           0          4.99         0           0
         Brand 4       0           0           0          4.99         0
         Other         0           0           0           0          4.99
```

We have been describing variables by their labels. While it is not necessary to look at it, the `&_trgind` macro variable name list that PROC TRANSREG creates for this problem is as follows:

```
%put &_trgind;
```

```
BrandBrand_1 BrandBrand_2 BrandBrand_3 BrandBrand_4 BrandOther
BrandBrand_1Price BrandBrand_2Price BrandBrand_3Price BrandBrand_4Price
BrandOtherPrice p1BrandBrand_1 p1BrandBrand_2 p1BrandBrand_3 p1BrandBrand_4
p1BrandOther p2BrandBrand_1 p2BrandBrand_2 p2BrandBrand_3 p2BrandBrand_4
p2BrandOther p3BrandBrand_1 p3BrandBrand_2 p3BrandBrand_3 p3BrandBrand_4
p3BrandOther p4BrandBrand_1 p4BrandBrand_2 p4BrandBrand_3 p4BrandBrand_4
p4BrandOther p5BrandBrand_1 p5BrandBrand_2 p5BrandBrand_3 p5BrandBrand_4
p5BrandOther
```

The analysis proceeds in exactly the same manner as before as follows:

```
proc phreg data=coded brief;
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;
```

The results are as follows:

```
                    Brand Choice Example, Multinomial Logit Model
                   Discrete Choice with Cross-Effects, Mother Logit

                                The PHREG Procedure

                                 Model Information

                         Data Set                WORK.CODED
                         Dependent Variable      c
                         Censoring Variable      c
                         Censoring Value(s)      2
                         Ties Handling           BRESLOW
                    Number of Observations Read       4000
                    Number of Observations Used       4000

            Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                       Number of       Number of          Chosen          Not
            Pattern      Choices    Alternatives    Alternatives       Chosen

                  1          800               5               1            4

                                   Convergence Status

                     Convergence criterion (GCONV=1E-8) satisfied.

                                 Model Fit Statistics

                                           Without           With
                    Criterion           Covariates     Covariates

                    -2 LOG L              2575.101       2349.325
                    AIC                   2575.101       2389.325
                    SBC                   2575.101       2483.018

                    Testing Global Null Hypothesis: BETA=0

                Test                 Chi-Square       DF     Pr > ChiSq

                Likelihood Ratio       225.7752       20         <.0001
                Score                  218.4500       20         <.0001
                Wald                   190.0257       20         <.0001

                     Multinomial Logit Parameter Estimates

                                 Parameter      Standard
                           DF     Estimate         Error    Chi-Square    Pr > ChiSq

    Brand 1                 1      1.24963       1.31259        0.9064        0.3411
    Brand 2                 1     -0.16269       1.38579        0.0138        0.9065
    Brand 3                 1     -3.90179       1.56511        6.2150        0.0127
    Brand 4                 1      2.49435       1.25537        3.9480        0.0469
    Other                   0            0             .             .             .
    Brand 1 Price           1      0.51056       0.13178       15.0096        0.0001
    Brand 2 Price           1     -0.04920       0.13411        0.1346        0.7137
    Brand 3 Price           1     -0.27594       0.15517        3.1623        0.0754
    Brand 4 Price           1      0.28951       0.12192        5.6389        0.0176
    Other Price             0            0             .             .             .
    Brand 1 on Brand 1      0            0             .             .             .
    Brand 1 on Brand 2      1      0.51651       0.13675       14.2653        0.0002
    Brand 1 on Brand 3      1      0.66122       0.15655       17.8397        <.0001
    Brand 1 on Brand 4      1      0.32806       0.12664        6.7105        0.0096
    Brand 1 on Other        0            0             .             .             .
```

```
Brand 2 on Brand 1     1       -0.39876       0.12832         9.6561         0.0019
Brand 2 on Brand 2     0              0             .              .              .
Brand 2 on Brand 3     1       -0.01755       0.15349         0.0131         0.9090
Brand 2 on Brand 4     1       -0.33802       0.12220         7.6512         0.0057
Brand 2 on Other       0              0             .              .              .
Brand 3 on Brand 1     1       -0.43868       0.13119        11.1823         0.0008
Brand 3 on Brand 2     1       -0.31541       0.13655         5.3356         0.0209
Brand 3 on Brand 3     0              0             .              .              .
Brand 3 on Brand 4     1       -0.54854       0.12528        19.1723         <.0001
Brand 3 on Other       0              0             .              .              .
Brand 4 on Brand 1     1        0.24398       0.12781         3.6443         0.0563
Brand 4 on Brand 2     1       -0.01214       0.13416         0.0082         0.9279
Brand 4 on Brand 3     1        0.40500       0.15285         7.0211         0.0081
Brand 4 on Brand 4     0              0             .              .              .
Brand 4 on Other       0              0             .              .              .
Other on Brand 1       0              0             .              .              .
Other on Brand 2       0              0             .              .              .
Other on Brand 3       0              0             .              .              .
Other on Brand 4       0              0             .              .              .
Other on Other         0              0             .              .              .
```

The results consist of:

- four nonzero brand effects and a zero for the constant alternative
- four nonzero alternative-specific price effects and a zero for the constant alternative
- $5 \times 5 = 25$ cross-effects, the number of alternatives squared, but only $(5-1) \times (5-2) = 12$ of them are nonzero (four brands not counting Other affecting each of the remaining three brands).
  - There are three cross-effects for the effect of Brand 1 on Brands 2, 3, and 4.
  - There are three cross-effects for the effect of Brand 2 on Brands 1, 3, and 4.
  - There are three cross-effects for the effect of Brand 3 on Brands 1, 2, and 4.
  - There are three cross-effects for the effect of Brand 4 on Brands 1, 2, and 3.

All coefficients for the constant (other) alternative are zero as are the cross-effects of a brand on itself.

The mother logit model is used to test for violations of IIA (independence from irrelevant alternatives). IIA means the odds of choosing alternative c_i over c_j do not depend on the other alternatives in the choice set. Ideally, this more general model will not significantly explain more variation in choice than the restricted models. Also, if IIA is satisfied, few if any of the cross-effect terms should be significantly different from zero. (See pages 452, 468, 674, and 679 for other discussions of IIA.) In this case, it appears that IIA is *not* satisfied (the data are artificial), so the more general mother logit model is needed. The chi-square statistic is $2424.812 - 2349.325 = 75.487$ with $20 - 8 = 12$ *df* ($p < 0.0001$).

You could eliminate some of the zero parameters by changing `zero=none` to `zero='Other'` and eliminating `p5` (`p&m`) from the model as follows:

```
proc transreg design data=price nozeroconstant norestoremissing;
   model class(brand / zero='Other' separators='' ' ') | identity(price)
         identity(p1-p4) * class(brand / zero='Other' separators='' ' on ') /
         lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   label price = 'Price'
         p1 = 'Brand 1' p2 = 'Brand 2' p3 = 'Brand 3'
         p4 = 'Brand 4';
   id subj set c;
   run;
```

You could also eliminate the brand by price effects and instead capture brand by price effects as the cross-effect of a variable on itself as follows:

```
proc transreg design data=price nozeroconstant norestoremissing;
   model class(brand / zero='Other' separators='' ' ')
         identity(p1-p4) * class(brand / zero='Other' separators='' ' on ') /
         lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   label price = 'Price'
         p1 = 'Brand 1' p2 = 'Brand 2' p3 = 'Brand 3'
         p4 = 'Brand 4';
   id subj set c;
   run;
```

In both cases, the analysis (not shown) is run in the usual manner. Except for the elimination of zero terms, and in the second case, the change to capture the price effects in the cross-effects, the results are identical.

Aggregating the Data

In all examples so far (except the last part of the last vacation example), the data set has been created for analysis with one stratum for each choice set and subject combination. Such data sets can be large. The data can also be arrayed with a frequency variable and each choice set forming a separate stratum. The following example illustrates how:

```
title 'Brand Choice Example, Multinomial Logit Model';
title2 'Aggregate Data';

%let m = 5;  /* Number of Brands in Each Choice Set */
             /* (including Other)                   */

proc format;
   value brand 1 = 'Brand 1' 2 = 'Brand 2' 3 = 'Brand 3'
               4 = 'Brand 4' 5 = 'Other';
   run;

data price2;
   array p[&m] p1-p&m; /* Prices for the Brands */
   array f[&m] f1-f&m; /* Frequency of Choice   */

   input p1-p&m f1-f&m;
   keep set price brand freq c p1-p&m;

   * Store choice set number to stratify;
   Set = _n_;

   do Brand = 1 to &m;

      Price = p[brand];

      * Output first choice: c=1, unchosen: c=2;
      Freq = f[brand]; c = 1; output;

      * Output number of times brand is not chosen.;
      freq = sum(of f1-f&m) - freq; c = 2; output;

      end;

format brand brand.;

datalines;
3.99 5.99 3.99 5.99 4.99   4 29 16 42  9
5.99 5.99 5.99 5.99 4.99  12 19 22 33 14
5.99 5.99 3.99 3.99 4.99  34 26  8 27  5
5.99 3.99 5.99 3.99 4.99  13 37 15 27  8
5.99 3.99 3.99 5.99 4.99  49  1  9 37  4
3.99 5.99 5.99 3.99 4.99  31 12  6 18 33
3.99 3.99 5.99 5.99 4.99  37 10  5 35 13
3.99 3.99 3.99 3.99 4.99  16 14  5 51 14
;

proc print data=price2(obs=10);
   var set c freq price brand;
   run;
```

The results are as follows:

```
                 Brand Choice Example, Multinomial Logit Model
                                Aggregate Data

                  Obs    Set    c    Freq    Price     Brand

                    1     1     1      4      3.99    Brand 1
                    2     1     2     96      3.99    Brand 1
                    3     1     1     29      5.99    Brand 2
                    4     1     2     71      5.99    Brand 2
                    5     1     1     16      3.99    Brand 3
                    6     1     2     84      3.99    Brand 3
                    7     1     1     42      5.99    Brand 4
                    8     1     2     58      5.99    Brand 4
                    9     1     1      9      4.99    Other
                   10     1     2     91      4.99    Other
```

This data set has 5 brands times 2 observations times 8 choice sets for a total of 80 observations, compared to $100 \times 5 \times 8 = 4000$ using the standard method. Two observations are created for each alternative within each choice set. The first contains the number of people who chose the alternative, and the second contains the number of people who did not choose the alternative.

To analyze the data, specify `strata Set` and `freq Freq`. The following steps code and analyze the data:

```
proc transreg design data=price2 nozeroconstant norestoremissing;
   model class(brand / zero=none) identity(price) / lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   label price = 'Price';
   id freq set c;
   run;


proc phreg data=coded;
   title2 'Discrete Choice with Common Price Effect, Aggregate Data';
   model c*c(2) = &_trgind / ties=breslow;
   strata set;
   freq freq;
   run;
```

These steps produced the following results:

```
                 Brand Choice Example, Multinomial Logit Model
            Discrete Choice with Common Price Effect, Aggregate Data

                              The PHREG Procedure

                               Model Information

                        Data Set                 WORK.CODED
                        Dependent Variable       c
                        Censoring Variable       c
                        Censoring Value(s)       2
                        Frequency Variable       Freq
                        Ties Handling            BRESLOW

                      Number of Observations Read          80
                      Number of Observations Used          80
                      Sum of Frequencies Read            4000
                      Sum of Frequencies Used            4000

          Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                                Number of          Chosen          Not
         Stratum     Set     Alternatives    Alternatives       Chosen

               1     1                500             100          400
               2     2                500             100          400
               3     3                500             100          400
               4     4                500             100          400
               5     5                500             100          400
               6     6                500             100          400
               7     7                500             100          400
               8     8                500             100          400
         ---------------------------------------------------------------
           Total                     4000             800         3200

                               Convergence Status

                    Convergence criterion (GCONV=1E-8) satisfied.

                              Model Fit Statistics

                                    Without           With
                      Criterion     Covariates     Covariates

                      -2 LOG L        9943.373       9793.486
                      AIC             9943.373       9803.486
                      SBC             9943.373       9826.909
```

```
                        Testing Global Null Hypothesis: BETA=0

                Test                 Chi-Square       DF     Pr > ChiSq

                Likelihood Ratio       149.8868        5         <.0001
                Score                  153.2328        5         <.0001
                Wald                   142.9002        5         <.0001

                         Multinomial Logit Parameter Estimates

                          Parameter      Standard
                    DF     Estimate         Error    Chi-Square    Pr > ChiSq

         Brand 1     1      0.66727       0.12305       29.4065        <.0001
         Brand 2     1      0.38503       0.12962        8.8235        0.0030
         Brand 3     1     -0.15955       0.14725        1.1740        0.2786
         Brand 4     1      0.98964       0.11720       71.2993        <.0001
         Other       0            0             .             .             .
         Price       1      0.14966       0.04406       11.5379        0.0007
```

The summary table is small with eight rows, one row per choice set. Each row represents 100 chosen alternatives and 400 unchosen. The 'Analysis of Maximum Likelihood Estimates' table exactly matches the one produced by the standard analysis. The –2 LOG L statistics are different than before: 9793.486 now compared to 2425.214 previously. This is because the data are arrayed in this example so that the partial likelihood of the proportional hazards model fit by PROC PHREG with the `ties=breslow` option is now proportional to—not identical to—the likelihood for the choice model. However, the Model Chi-Square statistics, *df*, and *p*-values are the same as before. The two corresponding pairs of –2 LOG L's differ by a constant $9943.373-2575.101 = 9793.486-2425.214 = 7368.272 = 2\times800\times\log(100)$. Since the χ^2 is the –2 LOG L without covariates minus –2 LOG L with covariates, the constants cancel and the χ^2 test is correct for both methods.

The technique of aggregating the data and using a frequency variable can be used for other models as well, for example, with brand by price effects as follows:

```
proc transreg design data=price2 nozeroconstant norestoremissing;
   model class(brand / zero=none separators='' ' ') |
         identity(price) / lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   label price = 'Price';
   id freq set c;
   run;

proc phreg data=coded;
   title2 'Discrete Choice with Brand by Price Effects, Aggregate Data';
   model c*c(2) = &_trgind / ties=breslow;
   strata set;
   freq freq;
   run;
```

This step produced the following results:

```
                 Brand Choice Example, Multinomial Logit Model
          Discrete Choice with Brand by Price Effects, Aggregate Data

                            The PHREG Procedure

                             Model Information

                      Data Set                WORK.CODED
                      Dependent Variable      c
                      Censoring Variable      c
                      Censoring Value(s)      2
                      Frequency Variable      Freq
                      Ties Handling           BRESLOW

                    Number of Observations Read          80
                    Number of Observations Used          80
                    Sum of Frequencies Read            4000
                    Sum of Frequencies Used            4000

        Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                                Number of          Chosen          Not
        Stratum    Set       Alternatives    Alternatives       Chosen

              1    1                  500             100          400
              2    2                  500             100          400
              3    3                  500             100          400
              4    4                  500             100          400
              5    5                  500             100          400
              6    6                  500             100          400
              7    7                  500             100          400
              8    8                  500             100          400
        ---------------------------------------------------------------
          Total                      4000             800         3200

                             Convergence Status

                 Convergence criterion (GCONV=1E-8) satisfied.

                           Model Fit Statistics

                                   Without           With
                    Criterion     Covariates     Covariates

                    -2 LOG L        9943.373       9793.084
                    AIC             9943.373       9809.084
                    SBC             9943.373       9846.561
                    Testing Global Null Hypothesis: BETA=0

            Test                 Chi-Square       DF     Pr > ChiSq

            Likelihood Ratio       150.2891        8         <.0001
            Score                  154.2562        8         <.0001
            Wald                   143.1425        8         <.0001

                     Multinomial Logit Parameter Estimates

                             Parameter      Standard
                     DF       Estimate         Error    Chi-Square    Pr > ChiSq

       Brand 1        1       -0.00972       0.43555        0.0005        0.9822
       Brand 2        1       -0.62230       0.48866        1.6217        0.2028
       Brand 3        1       -0.81250       0.60318        1.8145        0.1780
       Brand 4        1        0.31778       0.39549        0.6456        0.4217
       Other          0              0             .         .             .
       Brand 1 Price  1        0.13587       0.08259        2.7063        0.1000
       Brand 2 Price  1        0.20074       0.09210        4.7512        0.0293
       Brand 3 Price  1        0.13126       0.11487        1.3057        0.2532
       Brand 4 Price  1        0.13478       0.07504        3.2255        0.0725
       Other Price    0              0             .         .             .
```

The only thing that changes from the analysis with one stratum for each subject and choice set combination is the likelihood.

Previously, with one stratum per choice set within subject, we compared these models as follows: "The difference $2425.214 - 2424.812 = 0.402$ is distributed χ^2 with $8 - 5 = 3$ *df* and is not statistically significant." The difference between two $-2\log(\mathcal{L}_C)$'s equals the difference between two $-2\log(\mathcal{L}_B)$'s, since the constant terms $(800 \times \log(100))$ cancel, $9793.486 - 9793.084 = 2425.214 - 2424.812 = 0.402$.

Choice and Breslow Likelihood Comparison

This section explains why the –2 LOG L values differ by a constant with aggregate data versus individual data. It can be skipped by all but the most dedicated readers.

Consider the choice model with a common price slope. Let x_0 represent the price of the brand. Let x_1, x_2, x_3, and x_4 be indicator variables representing the choice of brands. Let $\mathbf{x} = (x_0\ x_1\ x_2\ x_3\ x_4)$ be the vector of alternative attributes. (A sixth element for 'Other' is omitted, since its parameter is always zero given the other brands.)

Consider the first choice set. There are five distinct vectors of alternative attributes
$\mathbf{x}_1 = (3.99\ 1\ 0\ 0\ 0)$ $\mathbf{x}_2 = (5.99\ 0\ 1\ 0\ 0)$ $\mathbf{x}_3 = (3.99\ 0\ 0\ 1\ 0)$ $\mathbf{x}_4 = (5.99\ 0\ 0\ 0\ 1)$
$\mathbf{x}_5 = (4.99\ 0\ 0\ 0\ 0)$

The vector $\mathbf{x}_2$, for example, represents choice of Brand 2, and $\mathbf{x}_5$ represents the choice of Other. One hundred individuals are asked to choose one of the $m = 5$ brands from each of the eight sets. Let f_1, f_2, f_3, f_4, and f_5 be the number of times each brand is chosen. For the first choice set, $f_1 = 4$, $f_2 = 29$, $f_3 = 16$, $f_4 = 42$, and $f_5 = 9$. Let N be the total frequency for each choice set, $N = \sum_{j=1}^{5} f_j = 100$. The likelihood L_1^C for the first choice set data is

$$L_1^C = \frac{\exp\left(\left(\sum_{j=1}^{5} f_j \mathbf{x}_j\right) \boldsymbol{\beta}\right)}{\left[\sum_{j=1}^{5} \exp(\mathbf{x}_j \boldsymbol{\beta})\right]^N}$$

The joint likelihood for all eight choice sets is the product of the likelihoods

$$\mathcal{L}_C = \prod_{k=1}^{8} L_k^C$$

The Breslow likelihood for this example, L_k^B, for the *kth* choice set, is the same as the likelihood for the choice model, except for a multiplicative constant.

$$L_k^C = N^N L_k^B = 100^{100} L_k^B$$

Therefore, the Breslow likelihood for all eight choice sets is

$$\mathcal{L}_B = \prod_{k=1}^{8} L_k^B = N^{-8N} \mathcal{L}_C = 100^{-800} \mathcal{L}_C$$

The two likelihoods are not exactly the same, because each choice set is designated as a separate stratum, instead of each choice set within each subject.

The log likelihood for the choice model is

$$\begin{aligned}
\log(\mathcal{L}_C) &= 800 \times \log(100) + \log(\mathcal{L}_B), \\
\log(\mathcal{L}_C) &= 800 \times \log(100) + (-0.5) \times 9793.486, \\
\log(\mathcal{L}_C) &= -1212.607
\end{aligned}$$

and $-2\log(\mathcal{L}_C) = 2425.214$, which matches the earlier output. However, it is usually not necessary to obtain this value.

Food Product Example with Asymmetry and Availability Cross-Effects

This example is based on the choice example from page 255. This example discusses the multinomial logit model, number of parameters, choosing the number of choice sets, designing the choice experiment, long design searches, examining the design, examining the subdesigns, examining the aliasing structure, blocking the design, testing the design before data collection, generating artificial data, processing the data, coding, cross-effects, availability, multinomial logit model results, modeling subject attributes, results, and interpretation.

Consider the problem of using a discrete choice model to study the effect of introducing a retail food product. This might be useful, for instance, to refine a marketing plan or to optimize a product prior to test market. A typical brand team has several concerns such as knowing the potential market share for the product, examining the source of volume, and providing guidance for pricing and promotions. The brand team might also want to know what brand attributes have competitive clout and want to identify competitive attributes to which they are vulnerable.

To develop this further, assume our client wishes to introduce a line extension in the category of frozen entrées. The client has one nationally branded competitor, a regional competitor in each of three regions, and a profusion of private label products at the grocery chain level. The product can come in two different forms: stove-top or microwaveable. The client believes that the private labels are very likely to mimic this line extension and to sell it at a lower price. The client suspects that this strategy on the part of private labels might work for the stove-top version but not for the microwaveable, where they have the edge on perceived quality. They also want to test the effect of a shelf talker that draws attention to their product.

The Multinomial Logit Model

This problem can be set up as a discrete choice model in which a respondent's choice among brands, given choice set C_a of available brands, corresponds to the brand with the highest utility. For each brand i, the utility U_i is the sum of a systematic component V_i and a random component e_i. The probability of choosing brand i from choice set C_a is therefore:

$$P(i|C_a) = P(U_i > \max(U_j)) = P(V_i + e_i > \max(V_j + e_j)) \ \ \forall \ \ (j \neq i) \in C_a$$

Assuming that the e_i follow an extreme value type I distribution, the conditional probabilities $P(i|C_a)$ can be found using the multinomial logit (MNL) formulation of McFadden (1974).

$$P(i|C_a) = \exp(V_i)/\sum_{j\in C_a} \exp(V_j)$$

One of the consequences of the MNL formulation is the property of independence from irrelevant alternatives (IIA). Under the assumption of IIA, all cross-effects are assumed to be equal, so that if a brand gains in utility, it draws share from all other brands in proportion to their current shares. Departures from IIA exist when certain subsets of brands are in more direct competition and tend to draw a disproportionate amount of share from each other than from other members in the category.

IIA is frequently described using a transportation example. Say you have three alternatives for getting to work: bicycle, car, or a blue bus. If a fourth alternative became available, a red bus, then according to IIA the red bus should draw riders from the other alternatives in proportion to their current usage. However, in this case, IIA is violated, and instead the red bus draws more riders from the blue bus than from car drivers and bicycle riders.

The mother logit formulation of McFadden (1974) can be used to capture departures from IIA. In a mother logit model, the utility for brand i is a function of both the attributes of brand i and the attributes of other brands. The effect of one brand's attributes on another is termed a cross-effect. In the case of designs in which only subsets C_a of the full shelf set C appear, the effect of the presence/absence of one brand on the utility of another is termed an *availability cross-effect.* (See pages 452, 459, 674, and 679 for other discussions of IIA.)

Set Up

In the frozen entrée example, there are five alternatives: the client's brand, the client's line extension, a national branded competitor, a regional brand and a private label brand. Several regional and private labels can be tested in each market, then aggregated for the final model. Note that the line extension is treated as a separate alternative rather than as a level of the client brand. This enables us to model the source of volume for the new entry and to quantify any cannibalization that occurs. Each brand is shown at either two or three price points. Additional price points are included so that quadratic models of price elasticity can be tested. The indicator for the presence or absence of a brand in the shelf set is coded using one level of the `Price` variable. The layout of factors and levels is given in the following table.

Factors and Levels

Alternative	Factor	Levels	Brand	Description
1	X1	4	Client	1.29, 1.69, 2.09 + absent
2	X2	4	Client Line Extension	1.39, 1.89, 2.39, + absent
	X3	2		microwave/stove-top
	X4	2		shelf talker yes/no
3	X5	3	Regional	1.99, 2.49 + absent
4	X6	3	Private Label	1.49, 2.29 absent
	X7	2		microwave/stove-top
5	X8	3	National	1.99 + 2.39 + absent

In addition to intercepts and main effects, we also require that all two-way interactions within alternatives be estimable: `x2*x3, x2*x4, x3*x4` for the line extension and `x6*x7` for private labels. This enables us to test for different price elasticities by form (stove-top versus microwaveable) and to see if the promotion works better combined with a low price or with different forms. Using a linear model for `x1-x8`, the total number of parameters including the intercept, all main effects, and two-way interactions with brand is 25. This assumes that price is treated as qualitative. The actual number of parameters in the choice model is larger than this because of the inclusion of cross-effects.

Using indicator variables to code availability, the systematic component of utility for brand i can be expressed as:

$$V_i = a_i + \sum_k (b_{ik} \times x_{ik}) + \sum_{j \neq i} z_j (d_{ij} + \sum_l (g_{ijl} \times x_{jl}))$$

where

a_i	= intercept for brand i
b_{ik}	= effect of attribute k for brand i, where $k = 1, .., K_i$
x_{ik}	= level of attribute k for brand i
d_{ij}	= availability cross-effect of brand j on brand i
z_j	= availability code = $\begin{cases} 1 & \text{if } j \in C_a, \\ 0 & \text{otherwise} \end{cases}$
g_{ijl}	= cross-effect of attribute l for brand j on brand i, where $l = 1, .., L_j$
x_{jl}	= level of attribute l for brand j.

The x_{ik} and x_{jl} could be expanded to include interaction and polynomial terms. In an availability design, each brand is present in only a fraction of the choice sets. The size of this fraction or subdesign is a function of the number of levels of the alternative-specific variable that is used to code availability (usually price). For instance, if price has three valid levels and a fourth zero level to indicate absence, then the brand appears in only three out of four runs. Following Lazari and Anderson (1994), the size of each subdesign determines how many model equations can be written for each brand in the discrete choice model. If X_i is the subdesign matrix corresponding to V_i, then each X_i must be full rank to ensure that the choice set design provides estimates for all parameters.

To create the design, a full-factorial candidate set is generated consisting of 3456 runs. It is then reduced to 2776 runs that contain between two and four brands so that the respondent is never required to compare more than four brands at a time. In the model specification, we designate all variables as classification variables and require that all main effects and two-way interactions within brands be estimable. The number of runs calculations are based on the number of parameters that we wish to estimate in the various subdesigns $\mathbf{X}_i$ of $\mathbf{X}$. Assuming that there is a None alternative used as a reference level, the numbers of parameters required for various alternatives are shown in the next table along with the sizes of the subdesigns (rounded down) for various numbers of runs. Parameters for quadratic price models are given in parentheses. Note that the effect of private label being in a microwaveable or stove-top form (stove/micro cross-effect) is an explicit parameter under the client line extension.

The subdesign sizes are computed by taking the floor of the number of runs from the marginal times the expected proportion of runs in which the alternative appears. For example, for the client brand, which has three prices and not available and 22 runs, floor($22 \times 3/4$) = 16; for the competitor and 32 runs, floor($32 \times 2/3$) = 21. The number of runs chosen is `n=26`. This number provides adequate degrees of freedom for the linear price model and also permits estimation of direct quadratic price effects. Estimating quadratic cross-effects for price requires 32 runs at the very least. Although the technique of using two-way interactions between nominal level variables usually guarantees that all direct and cross-effects are estimable, it is sometimes necessary and good practice to check the ranks of the subdesigns for more complex models (Lazari and Anderson 1994).

Parameters

Effect	Client	Client Line Extension	Regional	Private Label	Competitor
intercept	1	1	1	1	1
availability cross-effects	4	4	4	4	4
direct price effect	1 (2)	1 (2)	1	1	1
price cross-effects	4 (8)	4 (8)	4	4	4
stove versus microwave	-	1	-	1	-
stove/micro cross-effects	-	1	-	-	-
shelf talker	-	1	-	-	-
price*stove/microwave	-	1 (2)	-	1	-
price*shelf talker	-	1 (2)	-	-	-
stove/micro*shelf talker	-	1	-	-	-
Total	10 (15)	16 (23)	10	12	10
Subdesign size					
22 runs	16	16	14	14	14
26 runs	19	19	17	17	17
32 runs	24	24	21	21	21

Designing the Choice Experiment

This example originated with Kuhfeld, Tobias, and Garratt (1994), long before the %MktRuns macro existed. At least for now, we will skip the customary step of running the %MktRuns macro to suggest a design size and instead use the original size of 26 choice sets.

We use the %MktEx autocall macro to create the design. (All of the autocall macros used in this book are documented starting on page 803.) To recap, we want to make the design $2^3 3^3 4^2$ in 26 runs, and we want the following interactions to be estimable: x2*x3 x2*x4 x3*x4 x6*x7. Furthermore, there are restrictions on the design. Each of the price variables, x1, x2, x5, x6, and x8, has one level—the maximum level—that indicates the alternative is not available in the choice set. We use this to create choice sets with 2, 3, or 4 alternatives available. If (x1 < 4) then the first alternative is available, if (x2 < 4) then the second alternative is available, if (x5 < 3) then the third alternative is available, and so on. A Boolean term such as (x1 < 4) is one when true and zero otherwise. Hence,

```
((x1 < 4) + (x2 < 4) + (x5 < 3) + (x6 < 3) + (x8 < 3))
```

is the number of available alternatives. It is simply the sum of some 1's if available and 0's if not available.

We impose restrictions with the %MktEx macro by writing a macro, with IML statements, that quantifies the badness of each run (or in this case, each choice set). We do this so bad = 0 is good and values larger than zero are increasingly worse. We write our restrictions using an IML row vector x that contains the levels (integers beginning with 1) of each of the factors in the *ith* choice set, the one the macro is currently seeking to improve. The *jth* factor is x[j], or we can also use the factor names (for example, x1, x2). (See pages 604 and 1079 for other examples of restrictions.)

We must use IML logical operators, which do not have all of the same syntactical alternatives as DATA step operators:

Specify	For	Do Not Specify
=	equals	EQ
∧ = or ¬ =	not equals	NE
<	less than	LT
<=	less than or equal to	LE
>	greater than	GT
>=	greater than or equal to	GE
&	and	AND
|	or	OR
∧ or ¬	not	NOT
`a <= b & b <= c`	range check	`a <= b <= c`

To restrict the design, we must specify `restrictions=`*macro-name*, in this case `restrictions=resmac`, that names the macro that quantifies badness. The first statement counts up the number of available alternatives. The next two set the actual badness value. Note that the `else bad = 0` statement is not necessary since `bad` is automatically initialized to zero by the `%MktEx` macro. If the number available is less than two or greater than 4, then `bad` gets set to the absolute difference between the number available and 3. Hence, zero available corresponds to `bad = 3`, one available corresponds to `bad = 2`, two through four available corresponds to `bad = 0`, and five available corresponds to `bad = 2`. Do not just set `bad` to zero when everything is fine and one otherwise, because the macro needs to know that when it switches from zero available to one available, it is going in the right direction. For simple restrictions like this, it does not matter very much. However, for complicated sets of restrictions, it is critical that the *bad* variable is set to a count of the number of current restriction violations. The following steps create the design:[*]

```
title 'Consumer Food Product Example';

%macro resmac;
  navail = (x1 < 4) + (x2 < 4) + (x5 < 3) + (x6 < 3) + (x8 < 3);
  if (navail < 2) | (navail > 4) then bad = abs(navail - 3);
  else                                bad = 0;
  %mend;

%mktex(4 4 2 2 3 3 2 3,              /* all attrs of all alternatives       */
       n=26,                         /* number of choice sets               */
       interact=x2*x3 x2*x4 x3*x4 x6*x7, /* interactions                    */
       restrictions=resmac,          /* name of restrictions macro          */
       seed=377,                     /* random number seed                  */
       outr=sasuser.EntreeLinDes1)   /* randomized design stored permanently */
```

[*]Due to machine, SAS release, and macro differences, you might not get exactly the same design used in this book, but the differences should be slight.

The initial messages that the macro displays are as follows:

```
NOTE: Generating the fractional-factorial design, n=27.
NOTE: Generating the candidate set.
NOTE: Performing 60 searches of 2,776 candidates, full-factorial=3,456.
```

The orthogonal array initialization part of the coordinate-exchange algorithm iterations initializes the first 26 rows of a 27 run fractional-factorial design. This design has 13 three-level factors, ten of which are used to make $2^3 3^3 4^2$. The initial design is unbalanced and one row short of orthogonal, so we would expect other methods to be better for this problem. The macro also tells us that it is performing 60 PROC OPTEX searches of 2776 candidates, and that the full-factorial design has 3456 runs. The macro is searching the full-factorial design minus the excluded choice sets. Since the full-factorial design is not too large (less than 5000), and since there is no orthogonal array that is very good for this problem, this is the kind of problem where we expect the PROC OPTEX modified Fedorov algorithm (Fedorov 1972; Cook and Nachtsheim 1980) algorithm to work best. The macro chose 60 OPTEX iterations. In the fabric softener example, the macro did not try any OPTEX iterations, because it knew it could directly make a 100% *D*-efficient design. In the vacation examples, it ran the default minimum of 20 OPTEX iterations because the macro's heuristics concluded that OPTEX is probably not be the best approach for those problems. In this example, the macro's heuristics tried more iterations, since this is the kind of example where OPTEX works best.

Some of the output is as follows:

```
                      Consumer Food Product Example

                        Algorithm Search History

                             Current          Best
Design    Row,Col  D-Efficiency  D-Efficiency  Notes
----------------------------------------------------------
     1      Start       84.3176                Can
     1      2   1       84.3176       84.3176  Conforms
     1        End       84.3176

     2      Start       27.8626                Tab,Unb,Ran
     2      1   1       76.5332                Conforms
     2        End       80.4628
     .
     .
     .

    11      Start       24.5507                Tab,Ran
    11     26   1       78.6100                Conforms
    11        End       81.8604

    12      Start       26.3898                Ran,Mut,Ann
    12      1   1       67.0450                Conforms
    12        End       83.0114
     .
     .
     .
```

```
                  21      Start       45.9310                Ran,Mut,Ann
                  21     15   1       67.1046                Conforms
                  21        End       82.1657

      NOTE: Performing 600 searches of 2,776 candidates.

                            Consumer Food Product Example

                                Design Search History

                                  Current          Best
                Design    Row,Col  D-Efficiency  D-Efficiency  Notes
                ----------------------------------------------------------
                     0    Initial       84.3176       84.3176  Ini

                     1      Start       84.7548                Can
                     1      2   1       84.7548       84.7548  Conforms
                     1        End       84.7548

                            Consumer Food Product Example

                              Design Refinement History

                                  Current          Best
                Design    Row,Col  D-Efficiency  D-Efficiency  Notes
                ----------------------------------------------------------
                     0    Initial       84.7548       84.7548  Ini

                     1      Start       84.7548                Pre,Mut,Ann
                     1      2   1       82.6737                Conforms
                     1     14   1       84.7548       84.7548
                     1        End       82.6386
                     .
                     .
                     .

                     8      Start       84.7548                Pre,Mut,Ann
                     8      2   1       84.7548       84.7548  Conforms
                     8     14   1       84.7548       84.7548
                     8     21   2       84.7548       84.7548
                     8     12   3       84.7548       84.7548
                     8     12   6       84.7548       84.7548
                     8     18   1       84.7548       84.7548
                     8      2   2       84.7548       84.7548
                     8        End       84.7548

      NOTE: Stopping since it appears that no improvement is possible.
```

```
                          Consumer Food Product Example

                              The OPTEX Procedure

                            Class Level Information

                           Class  Levels    -Values-

                              x1       4    1 2 3 4
                              x2       4    1 2 3 4
                              x3       2    1 2
                              x4       2    1 2
                              x5       3    1 2 3
                              x6       3    1 2 3
                              x7       2    1 2

                          Consumer Food Product Example

                              The OPTEX Procedure

                                                                     Average
                                                                    Prediction
      Design                                                         Standard
      Number     D-Efficiency     A-Efficiency     G-Efficiency       Error
      ------------------------------------------------------------------------
           1         84.7548          71.1686          98.0583          0.9806
```

Design 1 (`Can`), which is created by the candidate-set search (using PROC OPTEX), has *D*-efficiency or 84.3176, and the macro confirms that the design conforms to our restrictions. The orthogonal array, unbalanced, and random initializations do not work as well. For each design, the macro iteration history states the *D*-efficiency for the initial design (27.8626 in design 2), the *D*-efficiency when the restrictions are met (76.5332, `Conforms`), and the *D*-efficiency for the final design (80.4628). The fully-random initialization tends to work a little better than the orthogonal array initialization for this problem, but not as well as PROC OPTEX. At the end of the algorithm search phase, the macro decides to use PROC OPTEX and performs 600 more searches, and it finds a design with 84.7548% *D*-efficiency. The design refinement step fails to improve on the best design. This step took 3.5 minutes.

Restrictions Formulated Using Actual Attribute Names and Levels

In the previous section, we constructed a macro to create a design, maximizing efficiency while forcing the design to conform to a set of restrictions. We provide `%MktEx` with a macro that it used to evaluate how well it is doing in imposing the restrictions. Internally, `%MktEx` stores its results in matrices, vectors, and scalars, with fixed names: `x1`, `x2`, `x3`, and so on. Levels of the factors are always positive integers starting with one. In practice, before you actually use the design, you will usually want to change these names and levels to something else that is more meaningful to your particular problem. In some cases, it might be more convenient to express your restrictions in terms of these more meaningful names and levels as well. To do that, you need to convert the `%MktEx` names and levels into your names and levels, for example, as follows:

```
%macro resmac;
   client    = {1.29 1.69 2.09 .}[x1];
   extension = {1.39 1.89 2.39 .}[x2];
   regional  = {1.99 2.49      .}[x5];
   private   = {1.49 2.29      .}[x6];
   national  = {1.99 2.39      .}[x8];

   navail    = (client  ^= .) + (extension ^= .) + (regional ^= .) +
               (private ^= .) + (national  ^= .);
   if (navail < 2) | (navail > 4) then bad = abs(navail - 3);
   else                                bad = 0;
   %mend;
```

The first line creates a scalar called `client`, that has one of four values: 1.29 when `x1` is 1, 1.69 when `x1` is 2, 2.09 when `x1` is 3, and . (missing) when `x1` is 4. Each of the first five assignment statements uses the `%MktEx` factor variables as an index into a constant vector of levels and stores the result in the attribute name so that restrictions can be posed in terms of the actual design problem. A literal vector consists of { `list` } and you can provide a subscript variable to index any vector, including a literal vector by specifying `[ variable ]`. Hence, the first five statements just convert `x1`, `x2`, `x5`, `x6`, and `x8` into names and levels that are familiar to the problem. Character vectors are allowed as well. The following step shows an alternative but equivalent formulation of the macro, treating the first two factors as numeric and the rest as character:

```
%macro resmac;
   client    = {1.29 1.69 2.09 .}[x1];
   extension = {1.39 1.89 2.39 .}[x2];
   regional  = {"$1.99" "$2.49" "NA"}[x5];
   private   = {"$1.49" "$2.29" "NA"}[x6];
   national  = {"$1.99" "$2.39" "NA"}[x8];

   navail    = (client ^= .) + (extension ^= .) + (regional ^= "NA") +
               (private ^= "NA") + (national  ^= "NA");
   if (navail < 2) | (navail > 4) then bad = abs(navail - 3);
   else                                bad = 0;
   %mend;
```

You cannot mix quoted character strings and unquoted numbers within a vector, however you can put any value in quotes. There is one extremely important caveat. Several names are used internally by `%MktEx` , and are available for you to look at when writing your restrictions. You must not change them. Do not use any of these names as names of anything you create when you write restrictions macros: `i`, `try`, `x`, `x1 x2` ..., `j1 j2` ..., `xmat`, `__pbad`, `__bbad`.

Macros like this are always be less efficient than macros based on the internal names since extra assignments are needed and the statements in the macro are evaluated **many** times. However, the slight loss of computer efficiency might be more than offset by the convenience of avoiding converting your restrictions to use internal names. It might not matter much for a problem such as this, but for complicated restrictions, it might be much easier to work with more informative names and levels. You can make the macro slightly more efficient, at a cost of a slightly more complicated looking restrictions macro, by using options=nox to suppress the creation of x1, x2, and so on, and instead make the mnemonic names and levels directly based on x. The macro must be changed, substituting x[1] for x1, x[2] for x2, and so on, as follows:

```
%macro resmac;
   client    = {1.29 1.69 2.09 .}[x[1]];
   extension = {1.39 1.89 2.39 .}[x[2]];
   regional  = {"$1.99" "$2.49" "NA"}[x[5]];
   private   = {"$1.49" "$2.29" "NA"}[x[6]];
   national  = {"$1.99" "$2.39" "NA"}[x[8]];

   navail    = (client ^= .) + (extension ^= .) + (regional ^= "NA") +
               (private ^= "NA") + (national  ^= "NA");
   if (navail < 2) | (navail > 4) then bad = abs(navail - 3);
   else                                bad = 0;
   %mend;

%mktex(4 4 2 2 3 3 2 3,             /* all attrs of all alternatives        */
       n=26,                        /* number of choice sets                */
       interact=x2*x3 x2*x4 x3*x4 x6*x7, /* interactions                    */
       optiter=0,                   /* no PROC OPTEX iterations             */
       tabiter=0,                   /* no OA initialization iterations      */
       maxdesigns=10,               /* make at most 10 designs              */
       options=nox,                 /* suppress x1, x2, ... creation        */
       restrictions=resmac,         /* name of restrictions macro           */
       seed=377)                    /* random number seed                   */
```

When You Have a Long Time to Search for an Efficient Design

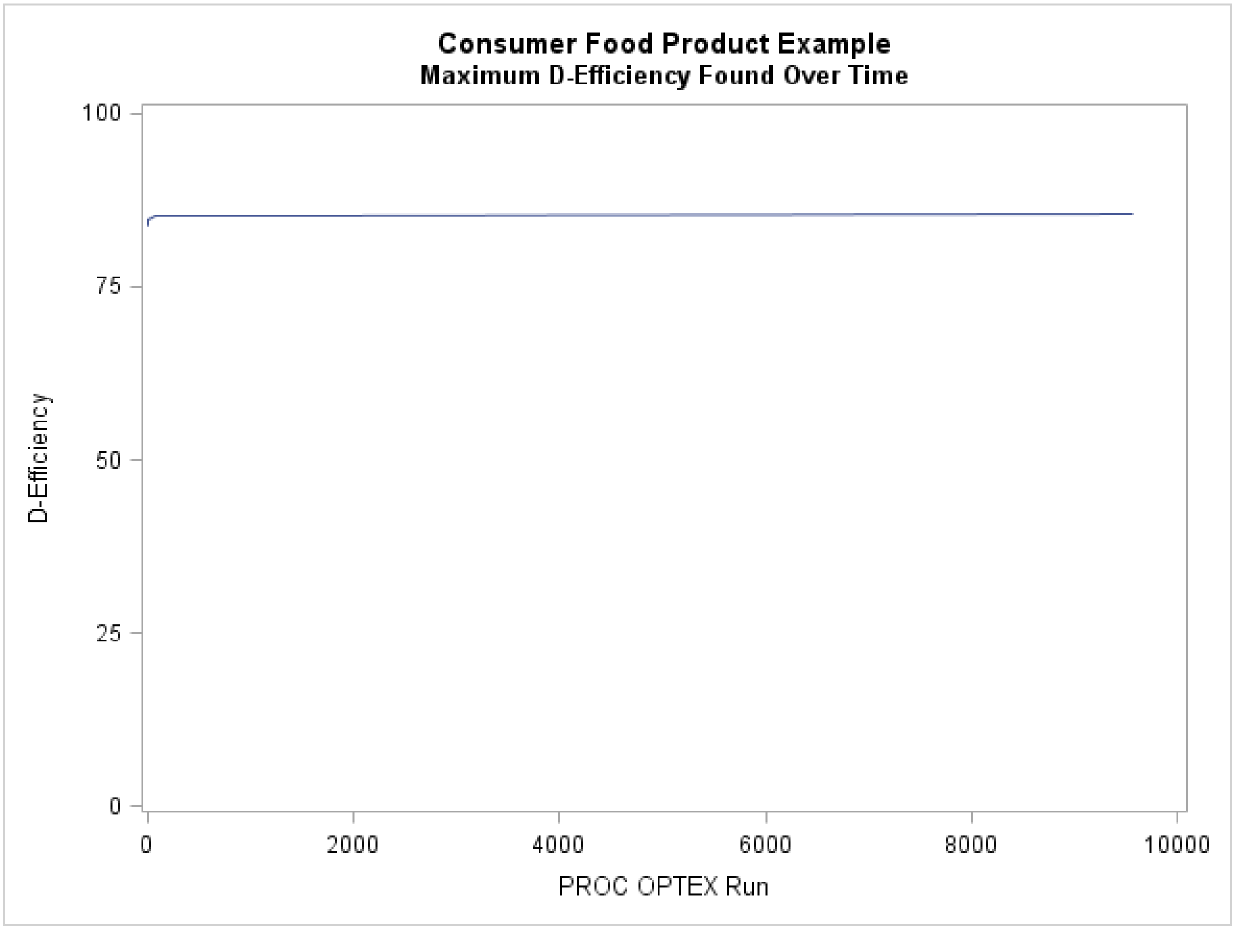
With a moderate sized candidate set such as this one (2000 to 6000 runs), we might be able to do better with more iterations. To test this, PROC OPTEX was run 10,000 times over the winter holiday vacation, from December 22 through January 2, creating a total of 200,000 designs, 20 designs on each try. (This was many years ago on computers that were much slower than the ones we have today.) The following table provides a summary of the results:

PROC OPTEX Run	*D*-Efficiency	Percent Improvement
1	83.8959	
2	83.9890	0.11%
3	84.3763	0.46%
6	84.7548	0.45%
84	85.1561	0.47%
1535	85.3298	0.20%
9576	85.3985	0.08%

This example is interesting, because it shows the diminishing value of increasing the number of iterations. Six minutes into the search, in the first six passes through PROC OPTEX ($6 \times 20 = 120$ total iterations), we found a design with reasonably good *D*-efficiency=84.7548. Over an hour into the search, with $(84 - 6) \times 20 = 1560$ more iterations, we get a small 0.47% increase in *D*-efficiency to 85.1561. About one day into the search, with $(1535 - 84) \times 20 = 29,020$ more iterations, we get another small 0.20% increase in *D*-efficiency, 85.3298. Finally, almost a week into the search, with $(9576 - 1535) \times 20 = 160,820$ more iterations, we get another small 0.08% increase in *D*-efficiency to 85.3985. Our overall improvement over the best design found in 120 iterations was 0.75952%, about three-quarters of a percent. These numbers will change with other problems and other seeds. However, as these results show, usually the first few iterations give you a good, efficient design, and usually, subsequent iterations give you slight improvements but with a cost of much greater run times. Next, we construct a plot of this table as follows:

```
data e;
   input n e;
   label n = 'PROC OPTEX Run' e = 'D-Efficiency';
   datalines;
    1  83.8959
    2  83.9890
    3  84.3763
    6  84.7548
   84  85.1561
 1535  85.3298
 9576  85.3985
;

proc sgplot data=e;
   title 'Consumer Food Product Example';
   title2 'Maximum D-Efficiency Found Over Time';
   series y=e x=n;
   yaxis values=(0 to 100 by 25);
   run;
```

The plot of maximum *D*-efficiency as a function of PROC OPTEX run number clearly shows that the gain in *D*-efficiency that comes from a large number of iterations is very slight.



If you have a lot of time to search for a good design, you can specify some of the time and maximum number of iteration parameters. Sometimes you might get lucky and find a better design. In this next example, `maxtime=300 300 60` is specified. This gives the macro up to 300 minutes for the algorithm search step, 300 minutes for the design search step, and 60 minutes for the refinement step. The option `maxiter=` increases the number iterations to 10000 for each of the three steps (or the maximum time). With this specification, you expect the macro to run overnight. See the macro documentation (starting on page 1017) for more iteration options. Note that you must increase the number of iterations and the maximum amount of time if you want the macro to run longer. With this specification, the macro performs 1800 OPTEX iterations initially (compared to 60 by default). The following steps create the restrictions macro and search for a design:

```
title 'Consumer Food Product Example';

%macro resmac;
  navail = (x1 < 4) + (x2 < 4) + (x5 < 3) + (x6 < 3) + (x8 < 3);
  if (navail < 2) | (navail > 4) then bad = abs(navail - 3);
  else                                bad = 0;
  %mend;

%mktex(4 4 2 2 3 3 2 3,            /* all attrs of all alternatives       */
       n=26,                       /* number of choice sets               */
       interact=x2*x3 x2*x4 x3*x4 x6*x7, /* interactions                 */
       restrictions=resmac,        /* name of restrictions macro          */
       seed=151,                   /* random number seed                  */
       maxtime=300 300 60,         /* max time for each major search phase */
       maxiter=10000)              /* max iterations for each phase       */
```

The results from this step are not shown.

Examining the Design

We can use the `%MktEval` macro to start to evaluate the design as follows:

```
%mkteval(data=sasuser.EntreeLinDes1)
```

The results are as follows:

```
                         Consumer Food Product Example
                   Canonical Correlations Between the Factors
             There are 4 Canonical Correlations Greater Than 0.316

             x1      x2      x3      x4      x5      x6      x7      x8

       x1    1       0.30    0.20    0.11    0.42    0.26    0.09    0.33
       x2    0.30    1       0.10    0.10    0.13    0.17    0.51    0.18
       x3    0.20    0.10    1       0.08    0.09    0.30    0       0.10
       x4    0.11    0.10    0.08    1       0.09    0.10    0       0.10
       x5    0.42    0.13    0.09    0.09    1       0.24    0.05    0.43
       x6    0.26    0.17    0.30    0.10    0.24    1       0.14    0.13
       x7    0.09    0.51    0       0       0.05    0.14    1       0.14
       x8    0.33    0.18    0.10    0.10    0.43    0.13    0.14    1
```

```
                Consumer Food Product Example
          Canonical Correlations > 0.316 Between the Factors
        There are 4 Canonical Correlations Greater Than 0.316

                                r    r Square

                 x2    x7    0.51      0.26
                 x5    x8    0.43      0.18
                 x1    x5    0.42      0.18
                 x1    x8    0.33      0.11

                Consumer Food Product Example
                    Summary of Frequencies
        There are 4 Canonical Correlations Greater Than 0.316
              * - Indicates Unequal Frequencies

                         Frequencies

 *    x1       7 8 6 5
 *    x2       6 7 7 6
      x3       13 13
      x4       13 13
 *    x5       9 8 9
 *    x6       7 10 9
 *    x7       12 14
 *    x8       7 9 10
 *    x1 x2    2 2 1 2 2 2 2 2 1 1 2 2 1 2 2 0
 *    x1 x3    3 4 4 4 4 2 2 3
 *    x1 x4    4 3 4 4 3 3 2 3
 *    x1 x5    4 2 1 2 1 5 2 2 2 1 3 1
 *    x1 x6    2 3 2 2 4 2 2 1 3 1 2 2
 *    x1 x7    3 4 4 4 3 3 2 3
 *    x1 x8    1 2 4 2 4 2 2 1 3 2 2 1
 *    x2 x3    3 3 3 4 4 3 3 3
 *    x2 x4    3 3 3 4 4 3 3 3
 *    x2 x5    2 2 2 3 2 2 2 2 3 2 2 2
 *    x2 x6    2 2 2 2 3 2 2 2 3 1 3 2
 *    x2 x7    1 5 4 3 2 5 5 1
 *    x2 x8    2 2 2 1 3 3 2 2 3 2 2 2
 *    x3 x4    7 6 6 7
 *    x3 x5    5 4 4 4 4 5
 *    x3 x6    2 5 6 5 5 3
 *    x3 x7    6 7 6 7
 *    x3 x8    4 4 5 3 5 5
 *    x4 x5    4 4 5 5 4 4
 *    x4 x6    4 5 4 3 5 5
 *    x4 x7    6 7 6 7
 *    x4 x8    4 4 5 3 5 5
```

```
*    x5 x6    2 4 3 2 2 4 3 4 2
*    x5 x7    4 5 4 4 4 5
*    x5 x8    1 2 6 4 2 2 2 5 2
*    x6 x7    3 4 4 6 5 4
*    x6 x8    2 2 3 2 4 4 3 3 3
*    x7 x8    4 4 4 3 5 6
     N-Way    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
              1 1 1 1 1 1 1
```

Some of the canonical correlations are bigger than we would like. They all involve attributes in different alternatives, so they should not pose huge problems. Still, they are large enough to make some researchers uncomfortable. The frequencies are pretty close to balanced. Perfect balance is not possible with 26 choice sets and this design. If we are willing to consider blocking the design, we might do better with more choice sets.

Designing the Choice Experiment, More Choice Sets

Let's run the `%MktRuns` macro to see what design size looks good. For now, we ignore the interactions and just specify main effects as follows:

```
%mktruns(4 4 2 2 3 3 2 3)
```

The results are as follows:

```
                Consumer Food Product Example

                       Design Summary

                    Number of
                    Levels       Frequency

                         2           3
                         3           3
                         4           2

                Consumer Food Product Example

         Saturated      = 16
         Full Factorial = 3,456
```

```
                    Some Reasonable                        Cannot Be
                       Design Sizes       Violations     Divided By

                                144 *              0
                                 72                1     16
                                 48                3      9
                                 96                3      9
                                192                3      9
                                 24                4      9 16
                                120                4      9 16
                                168                4      9 16
                                 36                7      8 16
                                108                7      8 16
                                 16 S             21      3  6  9 12

               * - 100% Efficient design can be made with the MktEx macro.
               S - Saturated Design - The smallest design that can be made.
                   Note that the saturated design is not one of the
                   recommended designs for this problem.  It is shown
                   to provide some context for the recommended sizes.

         n     Design                                                   Reference

       144     2 **118  3 **  4   4 **  2                            Orthogonal Array
       144     2 **115  3 **  4   4 **  3                            Orthogonal Array
       144     2 **113  3 ** 12   4 **  2                            Orthogonal Array
       144     2 **112  3 **  8   4 **  2   6 **  1                  Orthogonal Array
       144     2 **112  3 **  4   4 **  4                            Orthogonal Array
       144     2 **111  3 **  4   4 **  2   6 **  2                  Orthogonal Array
       144     2 **110  3 ** 12   4 **  3                            Orthogonal Array
           .
           .
           .
```

The smallest suggestion larger than 26 is 36. With this mix of factor levels, we would need 144 runs to get an orthogonal design (ignoring interactions), so we definitely want to stick with a nonorthogonal design. Balance is possible in 36 runs, but 36 cannot be divided by $2 \times 4 = 8$ and $4 \times 4 = 16$. With 36 runs, a blocking factor is required (2 blocks of 18 or 3 blocks of 12). We would like the shelf talker to appear in half of the choice sets within block, so with two blocks, we want the number of choice sets to be divisible by $2 \times 2 = 4$, and 36 can be divided by 4. Now let's specify the interactions as follows:

```
   %mktruns(4 4 2 2 3 3 2 3, interact=x2*x3 x2*x4 x3*x4 x6*x7)
```

The results are as follows:

```
                              Consumer Food Product Example

                                     Design Summary

                                 Number of
                                 Levels       Frequency

                                      2           3
                                      3           3
                                      4           2

                              Consumer Food Product Example

       Saturated      = 25
       Full Factorial = 3,456

       Some Reasonable                      Cannot Be
          Design Sizes       Violations     Divided By

                   144                2     32
                    96                5      9 18
                   192                5      9 18
                    48                7      9 18 32
                    72                9     16 32 48
                   216                9     16 32 48
                   120               14      9 16 18 32 48
                   168               14      9 16 18 32 48
                    36               25      8 16 24 32 48
                   108               25      8 16 24 32 48
                    25 S             61      2  3  4  6  8  9 12 16 18 24 32 48

              S - Saturated Design - The smallest design that can be made.
                  Note that the saturated design is not one of the
                  recommended designs for this problem.  It is shown
                  to provide some context for the recommended sizes.
```

Thirty-six runs is still in our list of possibilities, but now we see that not only can it not be divided by 8 and 16, it also cannot be divided by 24, 32, 48. We will try making a design in 36 runs and see how it looks.

In the previous try in 26 runs, the PROC OPTEX modified Fedorov algorithm worked best. There are two reasons why this probably happened. First, the full-factorial design is small enough to use as a candidate set. After imposing restrictions, the candidate set has 2,776 runs, and any size under 5000 or 10,000 is very manageable. Second, the design has interactions. The coordinate exchange algorithm by default considers only a single factor at a time, which is just one part of an interaction term. Modified Fedorov in contrast, considers exchanges involving all of the factors. For this problem, Modified Fedorov is invariably superior to the default coordinate-exchange algorithm. However, we can make coordinate exchange better, by having it perform multiple-column exchanges taking into account the interactions, just as we did in the vacation example on page 417. We use the `order=matrix=`*SAS-data-set* approach to looping over the columns of the design with the coordinate-exchange algorithm. In this case, coordinate exchange pairs columns 1, 5, and 8 with a randomly chosen column, it considers every possible triple in columns 2, 3, and 4, and it pairs columns 6 and 7 with a randomly chosen column. The following steps generate the order matrix and search for a design:

```
title 'Consumer Food Product Example';

%macro resmac;
  navail = (x1 < 4) + (x2 < 4) + (x5 < 3) + (x6 < 3) + (x8 < 3);
  if (navail < 2) | (navail > 4) then bad = abs(navail - 3);
  else                                bad = 0;
  %mend;

data mat;
   input a b c;
   datalines;
1 1 .
2 3 4
5 5 .
6 7 .
8 8 .
;

%mktex(4 4 2 2 3 3 2 3,            /* all attrs of all alternatives        */
       n=36,                       /* number of choice sets                */
       order=matrix=mat,           /* pairs/trips of cols to work on       */
       interact=x2*x3 x2*x4 x3*x4 x6*x7, /* interactions                   */
       restrictions=resmac,        /* name of restrictions macro           */
       seed=377,                   /* random number seed                   */
       outr=sasuser.EntreeLinDes2)  /* randomized design stored permanently */

%mkteval(data=sasuser.EntreeLinDes2)
```

A small part of the output from the `%MktEx` macro is as follows:

```
                         Consumer Food Product Example                        1

                            Algorithm Search History

                               Current          Best
              Design    Row,Col  D-Efficiency  D-Efficiency  Notes
              ----------------------------------------------------------
                   1      Start       94.0517                Can
                   1      2   1       94.0517       94.0517  Conforms
                   1        End       94.0517

                   .
                   .
                   .
```

```
                12      Start       71.6955                 Ran,Mut,Ann
                12       1   1      78.5418                 Conforms
                12      30   5      94.1433      94.1433
                12      33   5      94.1507      94.1507
                12      31   1      94.1532      94.1532
                12      23   6      94.1553      94.1553
                12        End       94.1553

                         Design Search History

                                    Current         Best
            Design    Row,Col  D-Efficiency  D-Efficiency  Notes
            ----------------------------------------------------------
                 0    Initial       94.1553       94.1553  Ini
                 .
                 .
                 .
                 3      Start       68.5288                 Ran,Mut,Ann
                 3      29   1      75.9029                 Conforms
                 3      22   5      94.1682      94.1682
                 3      34   5      94.1683      94.1683
                 3      35   6      94.2926      94.2926
                 3      16   8      94.3718      94.3718
                 3      24   6      94.3718      94.3718
                 3       9   1      94.4572      94.4572
                 3        End       94.2846
                 .
                 .
                 .

                     Consumer Food Product Example

                         The OPTEX Procedure

                                                                   Average
                                                                  Prediction
     Design                                                        Standard
     Number      D-Efficiency     A-Efficiency     G-Efficiency      Error
     ------------------------------------------------------------------------
          1         94.4571          88.7104          94.0740          0.8333
```

The `order=matrix=` option apparently helped. The coordinate exchange algorithm is in fact chosen over the modified Fedorov algorithm.

D-efficiency at 94.46% looks good. Part of the `%MktEval` results are as follows:

```
                       Consumer Food Product Example
                  Canonical Correlations Between the Factors
              There is 1 Canonical Correlation Greater Than 0.316

             x1      x2      x3      x4      x5      x6      x7      x8

       x1    1       0.13    0.10    0.11    0.11    0.17    0.10    0.12
       x2    0.13    1       0.12    0.08    0.23    0.39    0.06    0.18
       x3    0.10    0.12    1       0.06    0.10    0.04    0.00    0.10
       x4    0.11    0.08    0.06    1       0.07    0.07    0.06    0.18
       x5    0.11    0.23    0.10    0.07    1       0.13    0.04    0.15
       x6    0.17    0.39    0.04    0.07    0.13    1       0.04    0.13
       x7    0.10    0.06    0.00    0.06    0.04    0.04    1       0.04
       x8    0.12    0.18    0.10    0.18    0.15    0.13    0.04    1

                       Consumer Food Product Example
               Canonical Correlations > 0.316 Between the Factors
              There is 1 Canonical Correlation Greater Than 0.316


                                      r    r Square

                       x2    x6    0.39      0.15

                       Consumer Food Product Example
                           Summary of Frequencies
              There is 1 Canonical Correlation Greater Than 0.316
                        * - Indicates Unequal Frequencies

                          Frequencies

                  x1       9 9 9 9
              *   x2       8 9 10 9
              *   x3       19 17
                  x4       18 18
              *   x5       11 11 14
              *   x6       12 13 11
              *   x7       17 19
              *   x8       11 12 13
```

The correlations are better, although one is still not as good as we would like. The balance looks pretty good, however it would be nice if the balance, for example, in `x5` were better. It is often the case that improving balance requires some sacrifice of *D*-efficiency. We can run the macro again, this time specifying `balance=2`, which forces better balance. The specification of 2 allows the maximum frequency for a level in a factor to be no more than two greater than the minimum frequency. You should always specify `mintry=` with `balance=`. This allows `%MktEx` to at first increase *D*-efficiency while ignoring the balance restrictions. Then, after `mintry=`*m* rows have been processed, the balance restrictions are considered. Typically, you specify an expression that is a function of the number of rows for mintry=, for example, mintry=5 * n. The balance= option works best when its restrictions are imposed on a reasonably efficient design not an inefficient initial design.

This example also uses a somewhat more involved order=matrix data set. To understand why, you need to understand how the balance= option works. The %MktEx macro uses some of the following statements to impose balance:

```
__bbad = 1;
if try > &balancetry & j1 then do;
   acol = xmat[,j1];
   acol[i,] = x[,j1];
   acol = design(acol)[+,];
   __bbad = max(0, max(acol) - min(acol) - &balance);
   end;
```

It checks the balance restrictions based on the first column index, j1. If we are doing multiple exchanges, the exchanges in the second or subsequent columns could degrade the balance without it registering as a violation in the code above. For example, in the order=matrix=mat data set used previously, the last line is: 8 8 .. The column index j3 might change any of the columns and yet not register in the balance-checking code, because it is only looking at column 8. For this reason, we add eight more lines so the last thing the restrictions macro does in each row is check every column for the balance constraints. The following steps create and use the order matrix:

```
data mat;
   input a b c;
   datalines;
1 1 .
2 3 4
5 5 .
6 7 .
8 8 .
1 1 1
2 2 2
3 3 3
4 4 4
5 5 5
6 6 6
7 7 7
8 8 8
;

%mktex(4 4 2 2 3 3 2 3,            /* all attrs of all alternatives       */
       n=36,                        /* number of choice sets               */
       order=matrix=mat,            /* pairs/trips of cols to work on      */
       interact=x2*x3 x2*x4 x3*x4 x6*x7, /* interactions                   */
       restrictions=resmac,         /* name of restrictions macro          */
       seed=368,                    /* random number seed                  */
       outr=sasuser.EntreeLinDes3,  /* randomized design stored permanently */
       balance=2,                   /* require near but not perfect balance */
       mintry=5 * n)                /* ignore balance= for first 5 passes   */
```

The last part of the output from the `%MktEx` macro is as follows:

```
                            Consumer Food Product Example

                                The OPTEX Procedure

                                                                     Average
                                                                    Prediction
       Design                                                        Standard
       Number     D-Efficiency     A-Efficiency     G-Efficiency       Error
       ------------------------------------------------------------------------
            1         93.9552          87.8357          92.9627          0.8333
```

The *D*-efficiency looks good. It is a little lower than before, but not much. Next, we look at the canonical correlations and frequencies using the `%MktEval` macro as follows:

```
%mkteval(data=sasuser.EntreeLinDes3)
```

The first part of the output from the `%MktEval` macro is as follows:

```
                            Consumer Food Product Example
                      Canonical Correlations Between the Factors
                 There are 0 Canonical Correlations Greater Than 0.316

                 x1      x2      x3      x4      x5      x6      x7      x8

          x1     1       0.17    0.08    0.08    0.16    0.12    0.18    0.16
          x2     0.17    1       0.08    0.08    0.16    0.31    0.27    0.16
          x3     0.08    0.08    1       0.11    0.12    0.07    0       0.12
          x4     0.08    0.08    0.11    1       0.12    0.07    0       0.07
          x5     0.16    0.16    0.12    0.12    1       0.13    0.07    0.10
          x6     0.12    0.31    0.07    0.07    0.13    1       0.07    0.19
          x7     0.18    0.27    0       0       0.07    0.07    1       0.12
          x8     0.16    0.16    0.12    0.07    0.10    0.19    0.12    1
```

The canonical correlations look good. The last part of the output from the `%MktEval` macro is as follows:

```
                 Consumer Food Product Example
                    Summary of Frequencies
       There are 0 Canonical Correlations Greater Than 0.316
                * - Indicates Unequal Frequencies

                      Frequencies

 *    x1       9 8 9 10
 *    x2       8 9 10 9
      x3       18 18
      x4       18 18
 *    x5       12 11 13
 *    x6       11 12 13
      x7       18 18
 *    x8       11 12 13
 *    x1 x2    2 2 2 3 2 2 2 2 2 3 2 2 2 2 4 2
 *    x1 x3    5 4 4 4 4 5 5 5
 *    x1 x4    4 5 4 4 5 4 5 5
 *    x1 x5    3 3 3 2 2 4 3 3 3 4 3 3
 *    x1 x6    3 3 3 3 2 3 2 3 4 3 4 3
 *    x1 x7    4 5 5 3 5 4 4 6
 *    x1 x8    3 3 3 2 2 4 2 4 3 4 3 3
 *    x2 x3    4 4 4 5 5 5 5 4
 *    x2 x4    4 4 5 4 5 5 4 5
 *    x2 x5    3 3 2 3 2 4 3 3 4 3 3 3
 *    x2 x6    2 2 4 3 2 4 2 4 4 4 4 1
 *    x2 x7    2 6 5 4 6 4 5 4
 *    x2 x8    2 2 4 3 3 3 4 3 3 2 4 3
 *    x3 x4    8 10 10 8
 *    x3 x5    7 5 6 5 6 7
 *    x3 x6    5 6 7 6 6 6
      x3 x7    9 9 9 9
 *    x3 x8    6 5 7 5 7 6
 *    x4 x5    5 6 7 7 5 6
 *    x4 x6    6 6 6 5 6 7
      x4 x7    9 9 9 9
 *    x4 x8    6 6 6 5 6 7
 *    x5 x6    4 4 4 3 3 5 4 5 4
 *    x5 x7    6 6 6 5 6 7
 *    x5 x8    3 4 5 4 3 4 4 5 4
 *    x6 x7    5 6 6 6 7 6
 *    x6 x8    2 4 5 4 4 4 5 4 4
 *    x7 x8    5 7 6 6 5 7
      N-Way    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
               1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

This design looks much better. It is possible to get designs with better balance by specifying `balance=1`, however, since this gives `%MktEx` much less freedom, the `balance=1` option might cause *D*-efficiency to go down. Because `balance=1` is a tough restriction, we try this without `order=matrix` as follows:

```
%mktex(4 4 2 2 3 3 2 3,              /* all attrs of all alternatives       */
       n=36,                         /* number of choice sets               */
       interact=x2*x3 x2*x4 x3*x4 x6*x7, /* interactions                    */
       restrictions=resmac,          /* name of restrictions macro          */
       seed=472,                     /* random number seed                  */
       outr=sasuser.EntreeLinDes4,   /* randomized design stored permanently */
       balance=1,                    /* require near perfect balance        */
       mintry=5 * n)                 /* ignore balance= for first 5 passes   */

%mkteval(data=sasuser.EntreeLinDes4)
```

The *D*-efficiency, which is a lower than we saw previously, is as follows:

```
                          Consumer Food Product Example

                               The OPTEX Procedure

                                                                    Average
                                                                   Prediction
       Design                                                       Standard
       Number     D-Efficiency     A-Efficiency     G-Efficiency      Error
       ------------------------------------------------------------------------
            1        90.4983          79.9621          87.0176          0.8333
```

More troubling is the fact that the balance restrictions have increased the correlations between factors. The correlations are as follows:

```
                          Consumer Food Product Example
                      Canonical Correlations Between the Factors
                 There are 2 Canonical Correlations Greater Than 0.316

                 x1       x2       x3       x4       x5       x6       x7       x8

          x1     1        0.22     0.11     0.11     0.19     0.33     0.11     0.30
          x2     0.22     1        0.11     0.11     0.44     0.29     0.11     0
          x3     0.11     0.11     1        0        0.14     0        0        0.14
          x4     0.11     0.11     0        1        0.14     0        0.11     0.14
          x5     0.19     0.44     0.14     0.14     1        0.14     0        0.17
          x6     0.33     0.29     0        0        0.14     1        0.14     0
          x7     0.11     0.11     0        0.11     0        0.14     1        0.14
          x8     0.30     0        0.14     0.14     0.17     0        0.14     1
```

```
                    Consumer Food Product Example
               Canonical Correlations > 0.316 Between the Factors
              There are 2 Canonical Correlations Greater Than 0.316


                                       r    r Square

                       x2    x5    0.44      0.20
                       x1    x6    0.33      0.11
```

The rest of the results are as follows:

```
                    Consumer Food Product Example
                        Summary of Frequencies
              There are 2 Canonical Correlations Greater Than 0.316
                   * - Indicates Unequal Frequencies

                         Frequencies

                x1       9 9 9 9
                x2       9 9 9 9
                x3       18 18
                x4       18 18
                x5       12 12 12
                x6       12 12 12
                x7       18 18
                x8       12 12 12
```

The balance is perfect. Having balance in all of the factors is nice, but for this design, we only need to ensure that `x4`, the shelf talker factor is balanced, since we are dividing the design into two parts depending on whether the shelf talker is there or not. All things considered, it looks like the design that is created with `balance=2` is the best design for our situation. It is balanced in `x4`, it is either balanced or reasonably close to balanced in the other factors, and it has good D-efficiency and is reasonably close to orthogonal. If our design had not been balanced in `x4`, we could have tried again with a different seed, or we could have tried again with different values for `mintry=`. If the interactions had not been requested, we also could have switched it with another two-level factor, or added it after the fact by blocking (running the `%MktBlock` macro as if we are adding a blocking factor), or we could have used the `init=` option to constrain the factor to be balanced.

The `balance=` option in the `%MktEx` macro works by adding restrictions to the design. The approach it uses often works quite well, but sometimes it does not. Forcing balance gives the macro much less freedom in its search, and makes it easy for the macro to get stuck in suboptimal designs. If perfect balance is critical and there are no interactions or restrictions, you can also try the `%MktBal` macro.

Examining the Subdesigns

As we mentioned previously, "it is sometimes necessary and good practice to check the ranks of the subdesigns for more complex models (Lazari and Anderson 1994)." The next step shows one way to do that with PROC OPTEX. This is the only usage of PROC OPTEX in this book that is too specialized to be run from one of the `%Mkt` macros (because not all variables are designated as `class` variables). For convenience, we call PROC OPTEX from an ad hoc macro, since it must be run five times, once per alternative, with only a change in the `where` statement. We need to evaluate the design when the client's alternative is available `(x1 ne 4)`, when the client line extension alternative is available `(x2 ne 4)`, when the regional competitor is available `(x5 ne 3)`, when the private label competitor is available `(x6 ne 3)`, and when the national competitor is available `(x8 ne 3)`. We need to use a `model` statement that lists all of the main effects and interactions. We do not designate all of the variables in the `class` statement because we only have enough runs to consider linear price effects within each availability group. The statement `generate method=sequential initdesign=desv` specifies that we are evaluating the initial design `desv`, using the sequential algorithm, which ensures no swaps between the candidate set and the initial design. The other option of note here appears in the `class` statement, and that is `param=orthref`. This specifies an orthogonal parameterization of the effects that gives us a nice 0 to 100 scale for the *D*-efficiencies. The design is evaluated as follows:

```
%macro evaleff(where);
data desv / view=desv; set sasuser.EntreeLinDes3(where=(&where)); run;

proc optex data=desv;
   class x3 x4 x7 / param=orthref;
   model x1-x8 x2*x3 x2*x4 x3*x4 x6*x7;
   generate method=sequential initdesign=desv;
   run; quit;

%mkteval(data=desv)
%mend;

%evaleff(x1 ne 4)
%evaleff(x2 ne 4)
%evaleff(x5 ne 3)
%evaleff(x6 ne 3)
%evaleff(x8 ne 3)
```

Each step took just over two seconds. We hope to not see any efficiencies of zero, and we hope to not get the message `WARNING: Can't estimate model parameters in the final design.` Some of the results are as follows:

```
                       Consumer Food Product Example

                           The OPTEX Procedure

                                                                 Average
                                                               Prediction
Design                                                          Standard
Number     D-Efficiency     A-Efficiency     G-Efficiency        Error
---------------------------------------------------------------------------
     1          69.7007          61.6709          80.8872          0.7071

                                                                 Average
                                                               Prediction
Design                                                          Standard
Number     D-Efficiency     A-Efficiency     G-Efficiency        Error
---------------------------------------------------------------------------
     1          72.7841          64.9939          87.5576          0.6939

                                                                 Average
                                                               Prediction
Design                                                          Standard
Number     D-Efficiency     A-Efficiency     G-Efficiency        Error
---------------------------------------------------------------------------
     1          66.1876          50.8651          81.2554          0.7518

                                                                 Average
                                                               Prediction
Design                                                          Standard
Number     D-Efficiency     A-Efficiency     G-Efficiency        Error
---------------------------------------------------------------------------
     1          71.8655          59.8208          86.6281          0.7518

                                                                 Average
                                                               Prediction
Design                                                          Standard
Number     D-Efficiency     A-Efficiency     G-Efficiency        Error
---------------------------------------------------------------------------
     1          65.2313          50.1059          84.1610          0.7518
```

Examining the Aliasing Structure

It is also good to look at the aliasing structure of the design. We use PROC GLM to do this, so we must create a dependent variable. We use a constant `y=1`. The following PROC GLM step just checks the model to make sure none of the specified effects are aliased with each other:

```
data temp;
   set sasuser.EntreeLinDes3;
   y = 1;
   run;

proc glm data=temp;
   model y = x1-x8 x2*x3 x2*x4 x3*x4 x6*x7 / e aliasing;
   run; quit;
```

This step is not necessary since our *D*-efficiency value greater than zero already guarantees this. The results, ignoring the ANOVA and regression tables, which are not of interest, are as follows:

```
                                        Intercept
                                        x1
                                        x2
                                        x3
                                        x4
                                        x5
                                        x6
                                        x7
                                        x8
                                        x2*x3
                                        x2*x4
                                        x3*x4
                                        x6*x7
```

Each of these lines is a linear combination that is estimable. It is simply a list of the effects. Contrast this with a specification that includes all simple effects and two-way and three-way interactions. We specify the model of interest first, `x1-x8 x2*x3 x2*x4 x3*x4 x6*x7`, so all of those terms are listed first, then we specify all main effects and two-way and three-way interactions using the notation `x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8@3`. It is not a problem that some of the terms are both explicitly specified and also generated by the `x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8@3` list since PROC GLM automatically eliminates duplicate terms. The following step displays this more complicated aliasing structure:

```
proc glm data=temp;
   model y = x1-x8 x2*x3 x2*x4 x3*x4 x6*x7
             x1|x2|x3|x4|x5|x6|x7|x8@3 / e aliasing;
   run; quit;
```

The results are as follows:

```
    Intercept - 20.008*x4*x6 - 9.8483*x1*x4*x6 - 42.279*x2*x4*x6 - 9.0597*x3*x4*x6 +
    57.417*x5*x6 + 151.23*x1*x5*x6 + 186.61*x2*x5*x6 + 80.158*x3*x5*x6 +
    90.545*x4*x5*x6 - 50.89*x1*x7 + 4.2117*x2*x7 - 159.53*x1*x2*x7 + 12.566*x3*x7 -
    52.475*x1*x3*x7 + 43.269*x2*x3*x7 + 0.3801*x4*x7 - 71.5*x1*x4*x7 +
    36.725*x2*x4*x7 + 24.297*x3*x4*x7 + 21.563*x5*x7 - 27.16*x1*x5*x7 +
    75.528*x2*x5*x7 + 62.984*x3*x5*x7 + 39.224*x4*x5*x7 - 85.333*x1*x6*x7 -
    10.566*x2*x6*x7 + 15.818*x3*x6*x7 - 31.415*x4*x6*x7 + 123.51*x5*x6*x7 -
    24.144*x1*x8 + 6.6197*x2*x8 - 12.153*x1*x2*x8 - 38.1*x3*x8 - 133.06*x1*x3*x8 -
    135.02*x2*x3*x8 + 39.148*x4*x8 + 101.08*x1*x4*x8 + 149.27*x2*x4*x8 -
    15.467*x3*x4*x8 - 30.981*x5*x8 - 157.71*x1*x5*x8 - 130.69*x2*x5*x8 -
    107.69*x3*x5*x8 + 19.478*x4*x5*x8 - 40.116*x6*x8 - 116.84*x1*x6*x8 -
    61.852*x2*x6*x8 - 97.721*x3*x6*x8 - 23.772*x4*x6*x8 + 44.985*x5*x6*x8 -
    5.0186*x7*x8 - 171.5*x1*x7*x8 + 12.071*x2*x7*x8 - 2.9687*x3*x7*x8 +
    44.468*x4*x7*x8 + 8.5765*x5*x7*x8 - 52.648*x6*x7*x8
    x1 + 9.1371*x4*x6 + 7.8312*x1*x4*x6 + 17.618*x2*x4*x6 + 9.7563*x3*x4*x6 -
    21.745*x5*x6 - 69.803*x1*x5*x6 - 73.705*x2*x5*x6 - 39.359*x3*x5*x6 -
    25.304*x4*x5*x6 + 22.962*x1*x7 - 2.9296*x2*x7 + 71.792*x1*x2*x7 - 4.9586*x3*x7 +
    26.888*x1*x3*x7 - 12.562*x2*x3*x7 - 7.8969*x4*x7 + 11.379*x1*x4*x7 -
    35.377*x2*x4*x7 - 21.468*x3*x4*x7 - 12.723*x5*x7 + 10.604*x1*x5*x7 -
    43.808*x2*x5*x7 - 32.655*x3*x5*x7 - 32.497*x4*x5*x7 + 31.754*x1*x6*x7 +
    6.8554*x2*x6*x7 - 4.0467*x3*x6*x7 + 1.6149*x4*x6*x7 - 46.784*x5*x6*x7 -
    1.133*x1*x8 + 7.3858*x2*x8 + 2.0538*x1*x2*x8 + 4.336*x3*x8 + 3.3233*x1*x3*x8 +
    39.854*x2*x3*x8 - 5.3094*x4*x8 - 28.994*x1*x4*x8 - 5.5582*x2*x4*x8 +
    7.6916*x3*x4*x8 + 6.3495*x5*x8 + 15.979*x1*x5*x8 + 58.815*x2*x5*x8 +
    16.519*x3*x5*x8 + 11.175*x4*x5*x8 + 7.3054*x6*x8 + 13.278*x1*x6*x8 +
    29.443*x2*x6*x8 + 14.09*x3*x6*x8 + 18.767*x4*x6*x8 - 34.202*x5*x6*x8 +
    5.8152*x7*x8 + 65.231*x1*x7*x8 + 14.788*x2*x7*x8 - 3.885*x3*x7*x8 -
    15.536*x4*x7*x8 - 6.816*x5*x7*x8 + 18.202*x6*x7*x8
    .
    .
    .
```

Again, we have a list of linear combinations that are estimable. This shows that the `Intercept` cannot be estimated independently of the `x4*x6` interaction and a bunch of others including four-way though eight-way interactions which are not specified and hence not shown. Similarly, `x1` is confounded with a bunch of interactions, and so on. This is why we want to be estimable the two-way interactions between factors that are combined to create an alternative. We did not want something like `x2*x3`, the client-line extension's price and microwave/stove top interaction to be confounded with say another brand's price.

Blocking the Design

At 36 choice sets, this design is a bit large, so we can block it into two blocks of 18 choice sets. Within each block we want the shelf talker to be on half the time. The following step blocks the design:

```
%mktblock(data=sasuser.EntreeLinDes3, out=sasuser.EntreeLinDes,
          nblocks=2, seed=448)
```

The first attempt (not shown) produced a design where x4, the shelf talker did not occur equally often within each block. Changing the seed took care of the problem. The canonical correlations are as follows:

```
                          Consumer Food Product Example
                     Canonical Correlations Between the Factors
               There are 0 Canonical Correlations Greater Than 0.316

            Block    x1      x2      x3      x4      x5      x6      x7      x8

   Block    1        0.08    0.08    0       0       0.07    0.07    0       0.07
   x1       0.08     1       0.17    0.08    0.08    0.16    0.12    0.18    0.16
   x2       0.08     0.17    1       0.08    0.08    0.16    0.31    0.27    0.16
   x3       0        0.08    0.08    1       0.11    0.12    0.07    0       0.12
   x4       0        0.08    0.08    0.11    1       0.12    0.07    0       0.07
   x5       0.07     0.16    0.16    0.12    0.12    1       0.13    0.07    0.10
   x6       0.07     0.12    0.31    0.07    0.07    0.13    1       0.07    0.19
   x7       0        0.18    0.27    0       0       0.07    0.07    1       0.12
   x8       0.07     0.16    0.16    0.12    0.07    0.10    0.19    0.12    1
```

The blocking variable is not highly correlated with any of the factors. Some of the frequencies are as follows:

```
                          Consumer Food Product Example
                              Summary of Frequencies
               There are 0 Canonical Correlations Greater Than 0.316
                          * - Indicates Unequal Frequencies

                                 Frequencies

                   Block       18 18
               *   x1          9 8 9 10
               *   x2          8 9 10 9
                   x3          18 18
                   x4          18 18
               *   x5          12 11 13
               *   x6          11 12 13
                   x7          18 18
               *   x8          11 12 13
```

```
*    Block x1    5 5 4 4 4 4 5 5
*    Block x2    4 5 4 5 4 4 5 5
*    Block x3    8 10 9 9
     Block x4    9 9 9 9
*    Block x1    5 4 4 5 4 4 5 5
*    Block x2    4 4 5 5 4 5 5 4
     Block x3    9 9 9 9
     Block x4    9 9 9 9
*    Block x5    6 6 6 6 5 7
*    Block x6    5 6 7 6 6 6
     Block x7    9 9 9 9
*    Block x8    5 6 7 6 6 6
     .
     .
     .
```

The blocking variable is perfectly balanced, as it is guaranteed to be if the number of blocks divides the number of runs. Balance within blocks, that is the cross-tabulations of the factors with the blocking variable, looks good. The macro also displays canonical correlations within blocking variables. These can sometimes be quite high, even 1.0, but that is *not* a problem.[*] The design, as it is displayed by the `%MktBlock` macro, is as follows:

```
                    Consumer Food Product Example

Block     Run     x1     x2     x3     x4     x5     x6     x7     x8

  1         1      1      3      1      1      1      3      1      1
            2      3      1      2      2      1      3      2      1
            3      2      4      2      2      3      1      1      3
            4      4      3      1      2      2      2      1      1
            5      1      2      2      1      3      3      2      3
            6      4      3      1      1      3      2      2      3
            7      2      3      1      2      1      1      1      3
            8      1      1      2      1      2      2      2      3
            9      4      2      1      2      1      3      2      3
           10      3      1      1      1      1      1      2      3
           11      4      4      2      1      3      2      1      1
           12      4      3      2      2      3      3      1      2
           13      1      4      1      1      2      1      1      2
           14      2      2      1      1      2      3      1      1
           15      1      4      2      2      2      1      2      2
           16      3      4      2      1      1      2      2      2
           17      2      1      1      2      3      2      2      2
           18      3      2      2      2      2      3      1      2
```

[*]Ideally, each subject only makes one choice, since the choice model is based on this assumption (which is almost always ignored). As the number of blocks increases, the correlations mostly go to one, and ultimately are undefined when there is only one choice set per block.

```
                    Consumer Food Product Example

   Block     Run     x1     x2     x3     x4     x5     x6     x7     x8

     2         1      4      2      2      1      1      1      2      2
               2      1      3      2      2      3      2      2      1
               3      1      2      1      2      3      1      2      1
               4      4      1      1      1      2      3      2      1
               5      3      4      1      2      2      3      1      3
               6      2      4      1      1      3      1      2      3
               7      2      2      2      2      1      2      1      3
               8      2      1      2      1      3      3      1      2
               9      3      2      1      1      3      2      1      2
              10      1      4      1      2      1      2      1      2
              11      1      1      1      2      1      3      1      3
              12      3      2      2      1      3      1      1      1
              13      4      3      2      1      1      1      1      2
              14      3      3      2      1      2      2      1      3
              15      4      4      1      2      1      2      2      1
              16      2      3      2      1      2      3      2      1
              17      4      1      2      2      2      1      2      3
              18      3      3      1      2      3      3      2      2
```

The Final Design

The next steps create the final choice design, stored in `sasuser.EntreeChDes`, sorted by the blocking and shelf talker variable. We use the `%MktLab` macro to assign values, formats, and labels to the design. Previously, we have used the `%MktLab` macro to reassign factor names when we wanted something more descriptive than the default, `x1`, `x2`, and so on, and when we wanted to reassign the names of two m-level factors to minimize the problems associated with correlated factors. This time, we use the `%MktLab` macro primarily to deal with the asymmetry in the price factors. Recall our factor levels which are as follows:

Factors and Levels

Alternative	Factor	Levels	Brand	Description
1	X1	4	Client	1.29, 1.69, 2.09, absent
2	X2	4	Client Line Extension	1.39, 1.89, 2.39, absent
	X3	2		microwave/stove-top
	X4	2		shelf talker yes/no
3	X5	3	Regional	1.99, 2.49, absent
4	X6	3	Private Label	1.49, 2.29, absent
	X7	2		microwave/stove-top
5	X8	3	National	1.99 + 2.39, absent

The choice design needs a quantitative price attribute, made from all five of the linear price factors, that contains the prices of each of the alternatives. At this point, our factor `x1` contains 1, 2, 3, 4, and not 1.29, 1.69, 2.09, and absent, which is different from `x2` and from all of the other factors. A 1 in `x1` needs to become a price of 1.29 in the choice design, a 1 in `x2` needs to become a price of 1.39 in the choice design, a 1 in `x3` needs to become a price of 1.99 in the choice design, and so on. Before we use the `%MktRoll` macro to turn the linear arrangement into a choice design, we need to use the `%MktLab` macro to assign the actual prices to the price factors.

The `%MktLab` macro is like the `%MktRoll` macro in the sense that it can use as input a `key=` data set that contains the rules for customizing a design for our particular usage. In the `%MktRoll` macro, the `key=` data set provides the rules for turning a linear arrangement into a choice design. In contrast, in the `%MktLab` macro, the `key=` data set contains the rules for turning a linear arrangement into another linear arrangement, changing one or more of the following: factor names, factor levels, factor types (numeric to character), level formats, and factor labels.

We could use the `%MktLab` macro to change the names of the variables and their types, but we will not do that for this example. Ultimately, we use the `%MktRoll` macro to assign all of the price factors to a variable called `Price` and similarly provide meaningful names for all of the factors in the choice design, just as we have in previous examples. We could also change a variable like `x3` with values of 1 and 2 to something like `Stove` with values `'Stove'` and `'Micro'`. We will not do that because we want to make a design with a simple list of numeric factors, with simple names like `x1-x8` that we can run through the `%MktRoll` macro to get the final choice design. We assign formats and labels, so we can display the design in a meaningful way, but ultimately, our only goal at this step is to handle the price asymmetries by assigning the actual price values to the factors.

The `key=` data set contains the rules for customizing our design. The data set has as many rows as the maximum number of levels, in this case four. Each variable is one of the factors in the design, and the values are the factor levels that we want in the final design. The first factor, `x1`, is the price factor for the client brand. Its levels are 1.29, 1.69, and 2.09. In addition, one level is 'not available', which is flagged by the SAS special missing value `.N`. In order to read special missing values in an input data set, you must use the `missing` statement and name the expected missing values. The factor `x2` has the same structure as `x1`, but with different levels. The factor `x3` has two levels, hence the `key=` data set has missing values in the third and fourth row. Since the design has only 1's and 2's for `x3`, the missing values are never used. Notice that we are keeping `x3` as a numeric variable with values 1 and 2 using a format to supply the character levels 'micro' and 'stove'. The other factors are created in a similar fashion. By default, ordinary missing values `'.'` are not permitted as levels. By default, you can only use ordinary missing values as place holders for factors that have fewer levels than the maximum. If you want missing values in the levels, you must use one of the special missing values `.A`, `.B`, ..., `.Z`, and `._`* or the `cfill=` or `nfill=` options.

The `%MktLab` macro specification names the input SAS data set with the design and the key data set. By default, it creates an output SAS data set called `Final`. The data set is sorted by block and shelf talker and displayed in the following steps:

*Note that the `'.'` in `'.N'` is not typed in the data, nor is it typed in the `missing` statement. Furthermore, it does not appear in the displayed output. However, you need to type it if you ever refer to a special missing value in code: `if x1 eq .N then ...`.

```
proc format;
   value yn    1 = 'No'    2 = 'Talker';
   value micro 1 = 'Micro' 2 = 'Stove';
   run;

data key;
   missing N;
   input x1-x8;
   format x1 x2 x5 x6 x8 dollar5.2
          x4 yn. x3 x7 micro.;

   label x1 = 'Client Brand'
         x2 = 'Client Line Extension'
         x3 = 'Client Micro/Stove'
         x4 = 'Shelf Talker'

         x5 = 'Regional Brand'
         x6 = 'Private Label'
         x7 = 'Private Micro/Stove'
         x8 = 'National Competitor';

   datalines;
1.29 1.39 1 1 1.99 1.49 1 1.99
1.69 1.89 2 2 2.49 2.29 2 2.39
2.09 2.39 . . N    N    .    N
N    N    . . .    .    .    .
;

%mktlab(data=sasuser.EntreeLinDes, key=key)

proc sort out=sasuser.EntreeLinDesLab(drop=run); by block x4; run;

proc print label; id block x4; by block x4; run;
```

The `%MktLab` macro displays the variable mapping that it uses, old names followed by new names, as follows:

```
Variable Mapping:
   x1 : x1
   x2 : x2
   x3 : x3
   x4 : x4
   x5 : x5
   x6 : x6
   x7 : x7
   x8 : x8
```

In this case, none of the names change, but it is good to make sure that you have the expected correspondence.

The design is as follows:

```
                           Consumer Food Product Example

                          Client   Client                      Private
        Shelf   Client    Line     Micro/  Regional  Private  Micro/    National
 Block  Talker  Brand   Extension  Stove    Brand     Label    Stove   Competitor

   1    No      $1.29     $2.39    Micro    $1.99       N      Micro     $1.99
                $1.29     $1.89    Stove      N         N      Stove       N
                  N       $2.39    Micro      N       $2.29    Stove       N
                $1.29     $1.39    Stove    $2.49     $2.29    Stove       N
                $2.09     $1.39    Micro    $1.99     $1.49    Stove       N
                  N         N      Stove      N       $2.29    Micro     $1.99
                $1.29       N      Micro    $2.49     $1.49    Micro     $2.39
                $1.69     $1.89    Micro    $2.49       N      Micro     $1.99
                $2.09       N      Stove    $1.99     $2.29    Stove     $2.39

   1    Talker  $2.09     $1.39    Stove    $1.99       N      Stove     $1.99
                $1.69       N      Stove      N       $1.49    Micro       N
                  N       $2.39    Micro    $2.49     $2.29    Micro     $1.99
                $1.69     $2.39    Micro    $1.99     $1.49    Micro       N
                  N       $1.89    Micro    $1.99       N      Stove       N
                  N       $2.39    Stove      N         N      Micro     $2.39
                $1.29       N      Stove    $2.49     $1.49    Stove     $2.39
                $1.69     $1.39    Micro      N       $2.29    Stove     $2.39
                $2.09     $1.89    Stove    $2.49       N      Micro     $2.39

   2    No        N       $1.89    Stove    $1.99     $1.49    Stove     $2.39
                  N       $1.39    Micro    $2.49       N      Stove     $1.99
                $1.69       N      Micro      N       $1.49    Stove       N
                $1.69     $1.39    Stove      N         N      Micro     $2.39
                $2.09     $1.89    Micro      N       $2.29    Micro     $2.39
                $2.09     $1.89    Stove      N       $1.49    Micro     $1.99
                  N       $2.39    Stove    $1.99     $1.49    Micro     $2.39
                $2.09     $2.39    Stove    $2.49     $2.29    Micro       N
                $1.69     $2.39    Stove    $2.49       N      Stove     $1.99

   2    Talker  $1.29     $2.39    Stove      N       $2.29    Stove     $1.99
                $1.29     $1.89    Micro      N       $1.49    Stove     $1.99
                $2.09       N      Micro    $2.49       N      Micro       N
                $1.69     $1.89    Stove    $1.99     $2.29    Micro       N
                $1.29       N      Micro    $1.99     $2.29    Micro     $2.39
                $1.29     $1.39    Micro    $1.99       N      Micro       N
                  N         N      Micro    $1.99     $2.29    Stove     $1.99
                  N       $1.39    Stove    $2.49     $1.49    Stove       N
                $2.09     $2.39    Micro      N         N      Stove     $2.39
```

In contrast, the actual values without formats and labels are displayed by the following step:

```
proc print data=sasuser.EntreeLinDesLab; format _numeric_; run;
```

The results are as follows:

```
                         Consumer Food Product Example

      Obs     x1       x2      x3     x4     x5       x6      x7      x8      Block

        1    1.29     2.39     1      1     1.99      N       1     1.99        1
        2    1.29     1.89     2      1      N        N       2      N          1
        3     N       2.39     1      1      N       2.29     2      N          1
        4    1.29     1.39     2      1     2.49     2.29     2      N          1
        5    2.09     1.39     1      1     1.99     1.49     2      N          1
        6     N        N       2      1      N       2.29     1     1.99        1
        7    1.29      N       1      1     2.49     1.49     1     2.39        1
        8    1.69     1.89     1      1     2.49      N       1     1.99        1
        9    2.09      N       2      1     1.99     2.29     2     2.39        1
       10    2.09     1.39     2      2     1.99      N       2     1.99        1
       11    1.69      N       2      2      N       1.49     1      N          1
       12     N       2.39     1      2     2.49     2.29     1     1.99        1
       13    1.69     2.39     1      2     1.99     1.49     1      N          1
       14     N       1.89     1      2     1.99      N       2      N          1
       15     N       2.39     2      2      N        N       1     2.39        1
       16    1.29      N       2      2     2.49     1.49     2     2.39        1
       17    1.69     1.39     1      2      N       2.29     2     2.39        1
       18    2.09     1.89     2      2     2.49      N       1     2.39        1
       19     N       1.89     2      1     1.99     1.49     2     2.39        2
       20     N       1.39     1      1     2.49      N       2     1.99        2
       21    1.69      N       1      1      N       1.49     2      N          2
       22    1.69     1.39     2      1      N        N       1     2.39        2
       23    2.09     1.89     1      1      N       2.29     1     2.39        2
       24    2.09     1.89     2      1      N       1.49     1     1.99        2
       25     N       2.39     2      1     1.99     1.49     1     2.39        2
       26    2.09     2.39     2      1     2.49     2.29     1      N          2
       27    1.69     2.39     2      1     2.49      N       2     1.99        2
       28    1.29     2.39     2      2      N       2.29     2     1.99        2
       29    1.29     1.89     1      2      N       1.49     2     1.99        2
       30    2.09      N       1      2     2.49      N       1      N          2
       31    1.69     1.89     2      2     1.99     2.29     1      N          2
       32    1.29      N       1      2     1.99     2.29     1     2.39        2
       33    1.29     1.39     1      2     1.99      N       1      N          2
       34     N        N       1      2     1.99     2.29     2     1.99        2
       35     N       1.39     2      2     2.49     1.49     2      N          2
       36    2.09     2.39     1      2      N        N       2     2.39        2
```

One issue remains to be resolved regarding this design, and that concerns the role of the shelf talker when the client line extension is not available. The second part of each block of the design consists of choice sets in which the shelf talker is present and calls attention to the client line extension. However, in some of those choice sets, the client line extension is unavailable. This problem can be handled in several ways including the following:

- Rerun the design creation and evaluation programs excluding all choice sets with shelf talker present and client line extension unavailable. However, this requires changing the model because the excluded cell makes inestimable the interaction between client-line-extension price and shelf talker. Furthermore, the shelf talker variable will almost certainly no longer be balanced.
- Move the choice sets with client line extension unavailable to the no-shelf talker block and rerandomize. The shelf talker is then on for all of the last nine choice sets.
- Let the shelf talker go on and off as needed.
- Let the shelf talker call attention to a brand that happens to be out of stock. It is easy to imagine this happening in a real store.

Other options are available as well. No one approach is obviously superior to the alternatives. For this example, we take the latter approach and let the shelf talker be on even when the client line extension is not available. Note that if the shelf talker is turned off when the client line extension is not available then the design must be manually modified to reflect this fact.

Testing the Design Before Data Collection

This is a complicated design that is used to fit a complicated model with alternative-specific effects, price cross-effects, and availability cross-effects. Collecting data is time consuming and expensive. It is always good practice, and particularly when there are cross-effects, to make sure that the design works with the most complicated model that we anticipate fitting. Before we collect any data, we convert the linear arrangement to a choice design[*] and use the `%ChoicEff` macro to evaluate its efficiency for a multinomial logit model with both price and availability cross-effects.

For analysis, the design has four factors, `Brand`, `Price`, `Micro`, `Shelf`. We use the `%MktRoll` macro and a `key=` data set (although not the same one as before) to make the choice design. `Brand` is the alternative name; its values are directly read from the `key=Key` in-stream data. `Price` is an attribute whose values are constructed from the factors `x1`, `x2`, `x5`, `x6`, and `x8` in `sasuser.EntreeLinDesLab` data set. `Micro`, the microwave attribute, is constructed from `x3` for the client line extension and `x7` for the private label. `Shelf`, the shelf talker attribute, is created from `x4` for the extension. The `keep=` option in the `%MktRoll` macro is used to keep the original price factors in the design, since we need them for the price cross-effects. Normally, they are dropped. The following steps process the design and display a subset of the results:

[*]See page 67 for an explanation of the linear arrangement of a choice design versus the arrangement of a choice design that is more suitable for analysis.

```
data key;
   input Brand $ 1-10 (Price Micro Shelf) ($);
   datalines;
Client     x1 .  .
Extension  x2 x3 x4
Regional   x5 .  .
Private    x6 x7 .
National   x8 .  .
None       .  .  .
;

%mktroll(design=sasuser.EntreeLinDesLab, key=key, alt=brand, out=rolled,
         keep=x1 x2 x5 x6 x8)

proc print data=sasuser.EntreeLinDesLab(obs=2); run;

proc print data=rolled(obs=12);
   format price dollar5.2 shelf yn. micro micro.;
   id set; by set;
   run;
```

Consider the first two choice sets in the linear arrangement. They are as follows:

```
                             Consumer Food Product Example

       Obs      x1       x2     x3      x4      x5      x6     x7         x8   Block

         1   $1.29    $2.39    Micro    No    $1.99     N      Micro   $1.99     1
         2   $1.29    $1.89    Stove    No      N       N      Stove     N       1
```

When rolled out as a choice design, they are as follows:

```
                             Consumer Food Product Example

   Set    Brand        Price    Micro    Shelf        x1       x2       x5       x6       x8

    1     Client       $1.29        .        .    $1.29    $2.39    $1.99       N     $1.99
          Extension    $2.39    Micro    No       $1.29    $2.39    $1.99       N     $1.99
          Regional     $1.99        .        .    $1.29    $2.39    $1.99       N     $1.99
          Private        N      Micro        .    $1.29    $2.39    $1.99       N     $1.99
          National     $1.99        .        .    $1.29    $2.39    $1.99       N     $1.99
          None           .          .        .    $1.29    $2.39    $1.99       N     $1.99
   2   Client      $1.29      .        .  $1.29  $1.89   N    N    N
       Extension   $1.89   Stove   No     $1.29  $1.89   N    N    N
       Regional      N        .        .  $1.29  $1.89   N    N    N
       Private       N     Stove       .  $1.29  $1.89   N    N    N
       National      N        .        .  $1.29  $1.89   N    N    N
       None          .        .        .  $1.29  $1.89   N    N    N
```

Set 1, Alternative 1

`Brand`	=	’Client’	the brand for this alternative
`Price`	=	`x1` = $1.29	the price of this alternative
`Micro`	=	.	does not apply to this brand
`Shelf`	=	.	does not apply to this brand
`x1`	=	$1.29	the price of the client brand in this choice set
`x2`	=	$2.39	the price of the extension in this choice set
`x5`	=	$1.99	the price of the regional competitor in this choice set
`x6`	=	N	the private label unavailable in this choice set
`x8`	=	$1.99	national competitor unavailable in this choice set

Set 1, Alternative 2

`Brand`	=	’Extension’	the brand for this alternative
`Price`	=	`x2` = $2.39	the price of this alternative
`Micro`	=	Micro	Microwave version
`Shelf`	=	No	Shelf Talker, No
`x1`	=	$1.29	the price of the client brand in this choice set
`x2`	=	$2.39	the price of the extension in this choice set
`x5`	=	$1.99	the price of the regional competitor in this choice set
`x6`	=	N	the private label unavailable in this choice set
`x8`	=	$1.99	national competitor unavailable in this choice set

The factors `x1` through `x8` are used to make the price cross-effects. Notice that `x1` through `x8` are constant within each choice set. The variable `x1` is the price of alternative one, which is the same no matter which alternative it is stored with. The factors `x1` through `x8` are also used to make five other factors that are used to make the availability cross-effects. The following table shows how the prices are recoded for those factors:

`x1`	→	`a1`	`x2`	→	`a2`	`x5`	→	`a5`	`x6`	→	`a6`	`x8`	→	`a8`
1.29		1	1.39		1	1.99		1	1.49		1	1.99		1
1.69		1	1.89		1	2.49		1	2.29		1	2.39		1
2.09		1	2.39		1	N		−2	N		−2	N		−2
N		−3	N		−3									

This is a contrast coding. Within each factor, the coding sums to zero. Each availability factor has a coding that contrasts unavailable with the remaining available prices. When an alternative is unavailable, the `a` variable is set to minus the number of available price points. The coding for available alternatives is 1. A −3 is used for the first two alternatives that have three prices, and a −2 is used for the remaining alternatives that have two prices.

We need to do a few more things to this design before we are ready to use it. We need to convert the missings for when Micro and Shelf do not apply to 2 for 'Stove' and 1 for 'No'. We need to do the contrast coding for making the availability cross-effects. More is said about this after the code is shown. Since we are treating all of the price factors as a quantitative (not as class variables), we need to convert the missing prices to zero. Eventually, we also need to output just the alternatives that are available (those with a nonzero price and also the none alternative). For now, we just make a variable w that flags the available alternatives (w = 1). We can do this using a weight or flag variable: w = 1 means available and w = 0 means not available. We also need to assign labels and formats. The following DATA step does the processing:

```
data sasuser.EntreeChDes(drop=i);
   set rolled;
   array x[6] price x1 -- x8;
   array a[5] a1 a2 a5 a6 a8;
   if nmiss(micro) then micro = 2; /* stove if not a factor in alt      */
   if nmiss(shelf) then shelf = 1; /* not talker if not a factor in alt */

   a1 = -3 * nmiss(x1) + n(x1);    /* alt1: -3 - not avail, 1 - avail */
   a2 = -3 * nmiss(x2) + n(x2);    /* alt2: -3 - not avail, 1 - avail */
   a5 = -2 * nmiss(x5) + n(x5);    /* alt3: -2 - not avail, 1 - avail */
   a6 = -2 * nmiss(x6) + n(x6);    /* alt4: -2 - not avail, 1 - avail */
   a8 = -2 * nmiss(x8) + n(x8);    /* alt5: -2 - not avail, 1 - avail */
   i  = mod(_n_ - 1, 6) + 1;       /* alternative number              */
   if i le 5 then a[i] = 0;        /* 0 effect of an alt on itself    */

   do i = 1 to 6; if nmiss(x[i]) then x[i] = 0; end; /* missing price -> 0    */
   w = brand eq 'None' or price ne 0;                /* 1 - avail, 0 not avail*/
   format price dollar5.2 shelf yn. micro micro.;
   label x1 = 'CE, Client'         a1 = 'AE, Client'
         x2 = 'CE, Extension'      a2 = 'AE, Extension'
         x5 = 'CE, Regional'       a5 = 'AE, Regional'
         x6 = 'CE, Private'        a6 = 'AE, Private'
         x8 = 'CE, National'       a8 = 'AE, National';
   run;

proc print data=sasuser.EntreeChDes(obs=18); by set; id set; run;
```

The statements in the middle of the DATA step, from a1 = ... through if i ... create the variables that are used to make the availability effects. When alternative 1 is unavailable (x1 is missing), a1 is set to −3, otherwise a1 is set to 1; when alternative 2 is unavailable (x2 is missing), a2 is set to −3, otherwise a2 is set to 1; when alternative 3 is unavailable (x5 is missing), a5 is set to −3, otherwise a5 is set to 1; and so on. Each of these statements could have been written in if else form. Here, for example, is the first assignment statement rewritten: if nmiss(x1) then a1 = −3; else a1 = 1;. The variables x1, x2, x5, x6, and x8 are the five price factors, and the "a" factors use the same numbering scheme, although this is not a requirement. The if i and i = statements then set the variable to zero when the variable is used to construct the effect of an alternative on itself. For example, the first alternative is the client brand, so a1 in the first alternative is set to zero. The first three choice sets are as follows:

```
                           Consumer Food Product Example

   Set Brand     Price Micro Shelf    x1    x2    x5    x6    x8 a1 a2 a5 a6 a8 w

    1  Client    $1.29 Stove  No   $1.29 $2.39 $1.99 $0.00 $1.99  0  1  1 -2  1 1
       Extension $2.39 Micro  No   $1.29 $2.39 $1.99 $0.00 $1.99  1  0  1 -2  1 1
       Regional  $1.99 Stove  No   $1.29 $2.39 $1.99 $0.00 $1.99  1  1  0 -2  1 1
       Private   $0.00 Micro  No   $1.29 $2.39 $1.99 $0.00 $1.99  1  1  1  0  1 0
       National  $1.99 Stove  No   $1.29 $2.39 $1.99 $0.00 $1.99  1  1  1 -2  0 1
       None      $0.00 Stove  No   $1.29 $2.39 $1.99 $0.00 $1.99  1  1  1 -2  1 1

    2  Client    $1.29 Stove  No   $1.29 $1.89 $0.00 $0.00 $0.00  0  1 -2 -2 -2 1
       Extension $1.89 Stove  No   $1.29 $1.89 $0.00 $0.00 $0.00  1  0 -2 -2 -2 1
       Regional  $0.00 Stove  No   $1.29 $1.89 $0.00 $0.00 $0.00  1  1  0 -2 -2 0
       Private   $0.00 Stove  No   $1.29 $1.89 $0.00 $0.00 $0.00  1  1 -2  0 -2 0
       National  $0.00 Stove  No   $1.29 $1.89 $0.00 $0.00 $0.00  1  1 -2 -2  0 0
       None      $0.00 Stove  No   $1.29 $1.89 $0.00 $0.00 $0.00  1  1 -2 -2 -2 1

    3  Client    $0.00 Stove  No   $0.00 $2.39 $0.00 $2.29 $0.00  0  1 -2  1 -2 0
       Extension $2.39 Micro  No   $0.00 $2.39 $0.00 $2.29 $0.00 -3  0 -2  1 -2 1
       Regional  $0.00 Stove  No   $0.00 $2.39 $0.00 $2.29 $0.00 -3  1  0  1 -2 0
       Private   $2.29 Stove  No   $0.00 $2.39 $0.00 $2.29 $0.00 -3  1 -2  0 -2 1
       National  $0.00 Stove  No   $0.00 $2.39 $0.00 $2.29 $0.00 -3  1 -2  1  0 0
       None      $0.00 Stove  No   $0.00 $2.39 $0.00 $2.29 $0.00 -3  1 -2  1 -2 1
```

In the first choice set, for example, since alternative 1 is available, `a1` is 1, for all alternatives except the first, where `a1` is 0. Also in the first choice set, since alternative 4 is not available and there are two price levels, `a6` is -2 for all alternatives except the fourth, where `a6` is 0. In the third choice set, since alternative 1 is not available and there are three price levels, `a1` is -3, for all alternatives except the first, where `a2` is 0. In general, the coding stores a zero in the *ith* effect for the *ith* alternative, otherwise a 1 if the alternative is available, otherwise minus the number of price levels if the alternative is unavailable.

Now our choice design is done except for the final coding for the analysis. We can now use the `%ChoicEff` macro to evaluate our choice design. The complicated part of this is the model due to the alternative-specific price effects and the cross-effects. For now, let's concentrate on everything else. The following step provides some sample code, omitting for now the details of the model (indicated by `model= ...`):

```
%choiceff(data=sasuser.EntreeChDes, /* candidate set of choice sets        */
          init=sasuser.EntreeChDes(keep=set), /* select these sets         */
          intiter=0,                /* evaluate without internal iterations */
          model= ...,               /* model specification skipped for now  */
          nsets=36,                 /* number of choice sets                */
          nalts=6,                  /* number of alternatives               */
          weight=w,                 /* weight to handle unavailable alts    */
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

The way you check the efficiency of a design like this is to first name it in the `data=` option. This is the candidate set that contains all of the choice sets that we will consider. In addition, the same design is named in the `init=` option. The full specification is `init=sasuser.EntreeChDes(keep=set)`. Just the variable `Set` is kept. It is used to bring in just the indicated choice sets from the `data=` design, which in this case is all of them. The option `nsets=36` specifies the number of choice sets, and `nalts=6` specifies the number of alternatives. This macro requires a constant number of alternatives in each choice set for ease of data management. However, not all of the alternatives have to be used. In this case, we have an availability study. We need to keep the unavailable alternatives in the design for this step, but we do not want them to contribute to the analysis, so we specify a weight variable with `weight=w` and flag the available alternatives with `w=1` and the unavailable alternatives with `w=0`. The option `beta=zero` specifies that we are assuming for design evaluation purpose all zero betas. We can specify other values and get other results for the variances and standard errors. Finally, we specify `intiter=0` which specifies zero internal iterations. We use zero internal iterations when we want to evaluate an initial design, but not attempt to improve it. The actual specification we use, complete with the model specification, is shown in the following step:

```
%choiceff(data=sasuser.EntreeChDes, /* candidate set of choice sets        */
          init=sasuser.EntreeChDes(keep=set), /* select these sets          */
          intiter=0,                /* evaluate without internal iterations */
                                    /* cross-effects model                 */
                                    /* zero='None' - 'None' level is ref lev*/
          model=class(brand / zero='None')
                                    /* use blank sep to build interact terms*/
                class(brand / zero='None' separators='' ' ') *
                   identity(price)
                                    /* lpr=5 0 uses 5 var name chars in     */
                                    /* label for shelf and 0 for micro      */
                                    /* 'No' is ref level for shelf          */
                                    /* 'Stove' is ref level for micro       */
                class(shelf micro / lprefix=5 0 zero='No' 'Stove')
                                    /* ' on ' is used to build interact term*/
                identity(x1 x2 x5 x6 x8) *
                   class(brand / zero='None' separators=' ' ' on ')
                identity(a1 a2 a5 a6 a8) *
                   class(brand / zero='None' separators=' ' ' on ') /
                lprefix=0           /* lpr=0 labels are created from just   */
                                    /*    levels unless overridden above    */
                order=data,         /* order=data - do not sort levels      */

          nsets=36,                 /* number of choice sets                */
          nalts=6,                  /* number of alternatives               */
          weight=w,                 /* weight to handle unavailable alts    */
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

The specification `class(brand / zero='None')` specifies the brand effects. This specification creates indicator variables for brand with the constant alternative being the reference brand. The option `zero='None'` ensures that the reference level is `'None'` instead of the default last sorted level (`'Regional'`). Indicator variables are created for the brands Client, Extension, Regional, Private, and National, but not None. The `zero='None'` option, like `zero='Home'` and other `zero=`*'literal-string'* options we have used in previous examples, names the actual formatted value of the `class` variable that should be excluded from the coded variables because the coefficient is zero. Do not confuse `zero=none` and `zero='None'`. The `zero=none` option specifies that you want all indicator variables to be created, even including one for the last level. In contrast, the option `zero='None'` (or `zero=` any quoted string) names a specific formatted value, in this case `'None'`, for which indicator variables are not to be created. See page 78 for more information about the `zero=` option.

The specification `class(brand / ...) * identity(price)` creates the alternative-specific price effects. They are specified as an interaction between a categorical variable `Brand` and a quantitative attribute `Price`. The `separators='' ' '` option in the `class` specification specifies the separators that are used to construct the labels for the main effect and interaction terms. The main-effects separator, which is the first `separators=` value, `''`, is ignored since `lprefix=0`. Specifying `' '` as the second value creates labels of the form *brand-blank-price* instead of the default *brand-blank-asterisk-blank-price*.

The specification `class(shelf micro / ...)` names the shelf talker and microwave variables as categorical variables and creates indicator variables for the `'Talker'` category, not the `'No'` category and the `'Micro'` category not the `'Stove'` category. In `zero='No' 'Stove'`, the `'No'` applies to the first variable, `Shelf` and the second value, `'Stove'`, applies to second variable, `Micro`.

The specification `identity(x1 x2 x5 x6 x8) * class(brand / ...)` creates the linear price cross-effects. The `separators=` option is specified with a second value of `' on '` to create cross-effect labels like `'Client on Extension'`. The specification `identity(a1 a2 a5 a6 a8) * class(brand / ...)` creates the availability cross-effects. Note that the order of the transformation specification is important. Make sure you specify `identity` followed by `class` in order to get the right labels. More is said about cross-effects when we look at the actual coded values in the next few pages.

PROC TRANSREG produces the following warning:

```
WARNING: This usage of * sets one group's slope to zero.  Specify |
         to allow all slopes and intercepts to vary.  Alternatively,
         specify CLASS(vars) * identity(vars) identity(vars) for
         separate within group functions and a common intercept.
         This is a change from Version 6.
```

This is because `class` is interacted with `identity` using the asterisk instead of the vertical bar. In a linear model, this might be a sign of a coding error, so the procedure displays a warning. If you get this warning while coding a choice model specifying `zero=`*'constant-alternative-level'*, you can safely ignore it. Still, it is always good to display one or more coded choice sets to check the coding as we will do later. The last part of the output from the `%ChoicEff` macro is as follows:

```
                         Consumer Food Product Example

                                 Final Results

                          Design                    1
                          Choice Sets              36
                          Alternatives              6
                          Parameters               52
                          Maximum Parameters      180
                          D-Efficiency              0
                          D-Error                   .
                          Consumer Food Product Example

                                                                         Standard
  n   Variable Name        Label                         Variance   DF      Error

  1   BrandClient          Client                          69.807    1     8.3551
  2   BrandExtension       Extension                       75.688    1     8.6999
  3   BrandRegional        Regional                       121.147    1    11.0067
  4   BrandPrivate         Private                        104.058    1    10.2009
  5   BrandNational        National                       110.456    1    10.5098
  6   BrandClientPrice     Client Price                     3.255    1     1.8042
  7   BrandExtensionPrice  Extension Price                  2.233    1     1.4942
  8   BrandRegionalPrice   Regional Price                   6.599    1     2.5688
  9   BrandPrivatePrice    Private Price                    2.604    1     1.6138
 10   BrandNationalPrice   National Price                  11.071    1     3.3273
 11   ShelfTalker          Shelf Talker                     0.928    1     0.9636
 12   MicroMicro           Micro                            0.562    1     0.7493
 13   x1BrandClient        CE, Client on Client              .       0      .
 14   x1BrandExtension     CE, Client on Extension          4.689    1     2.1655
 15   x1BrandRegional      CE, Client on Regional           4.462    1     2.1124
 16   x1BrandPrivate       CE, Client on Private            5.627    1     2.3720
 17   x1BrandNational      CE, Client on National           5.374    1     2.3182
 18   x2BrandClient        CE, Extension on Client          3.040    1     1.7435
 19   x2BrandExtension     CE, Extension on Extension        .       0      .
 20   x2BrandRegional      CE, Extension on Regional        3.038    1     1.7431
 21   x2BrandPrivate       CE, Extension on Private         3.666    1     1.9146
 22   x2BrandNational      CE, Extension on National        3.130    1     1.7691
 23   x5BrandClient        CE, Regional on Client           8.961    1     2.9935
 24   x5BrandExtension     CE, Regional on Extension        9.824    1     3.1343
 25   x5BrandRegional      CE, Regional on Regional          .       0      .
 26   x5BrandPrivate       CE, Regional on Private         10.496    1     3.2398
 27   x5BrandNational      CE, Regional on National        10.360    1     3.2188
 28   x6BrandClient        CE, Private on Client            3.965    1     1.9912
 29   x6BrandExtension     CE, Private on Extension         4.195    1     2.0482
 30   x6BrandRegional      CE, Private on Regional          4.429    1     2.1046
 31   x6BrandPrivate       CE, Private on Private            .       0      .
 32   x6BrandNational      CE, Private on National          4.453    1     2.1102
 33   x8BrandClient        CE, National on Client          18.098    1     4.2541
 34   x8BrandExtension     CE, National on Extension       16.311    1     4.0387
 35   x8BrandRegional      CE, National on Regional        22.372    1     4.7299
 36   x8BrandPrivate       CE, National on Private         18.271    1     4.2745
 37   x8BrandNational      CE, National on National          .       0      .
 38   a1BrandClient        AE, Client on Client              .       0      .
 39   a1BrandExtension     AE, Client on Extension          0.981    1     0.9904
 40   a1BrandRegional      AE, Client on Regional           0.892    1     0.9447
 41   a1BrandPrivate       AE, Client on Private            1.071    1     1.0347
 42   a1BrandNational      AE, Client on National           1.031    1     1.0155
```

```
  43   a2BrandClient         AE, Extension on Client         0.766    1     0.8755
  44   a2BrandExtension      AE, Extension on Extension        .      0       .
  45   a2BrandRegional       AE, Extension on Regional       0.880    1     0.9381
  46   a2BrandPrivate        AE, Extension on Private        0.990    1     0.9952
  47   a2BrandNational       AE, Extension on National       0.999    1     0.9995

  48   a5BrandClient         AE, Regional on Client          5.128    1     2.2644
  49   a5BrandExtension      AE, Regional on Extension       5.530    1     2.3516
  50   a5BrandRegional       AE, Regional on Regional          .      0       .
  51   a5BrandPrivate        AE, Regional on Private         5.860    1     2.4208
  52   a5BrandNational       AE, Regional on National        5.887    1     2.4263

  53   a6BrandClient         AE, Private on Client           1.796    1     1.3402
  54   a6BrandExtension      AE, Private on Extension        1.843    1     1.3577
  55   a6BrandRegional       AE, Private on Regional         2.116    1     1.4547
  56   a6BrandPrivate        AE, Private on Private            .      0       .
  57   a6BrandNational       AE, Private on National         1.964    1     1.4015

  58   a8BrandClient         AE, National on Client         10.135    1     3.1836
  59   a8BrandExtension      AE, National on Extension       8.720    1     2.9529
  60   a8BrandRegional       AE, National on Regional       12.021    1     3.4671
  61   a8BrandPrivate        AE, National on Private        10.188    1     3.1919
  62   a8BrandNational       AE, National on National          .      0       .
                                                                     ==
                                                                     52
```

First we see estimable brand effects for each of the five brands, excluding the constant alternative `'None'`. Next, we see quantitative alternative-specific price effects for each of the brands. The next two effects that are single *df* effects for the shelf talker and the microwave option. Then we see five sets of linear price cross-effects (those whose label begins with "CE"), each consisting of four effects of a brand on another brand, plus one more zero *df* cross-effect of a brand on itself. The zero *df* and missing variances and standard errors are correct since the cross-effect of an alternative on itself is perfectly aliased with its alternative-specific price effect. After that we see five sets of availability cross-effects (those whose label begins with "AE"), each consisting of four effects of a brand on another brand, plus one more zero *df* cross-effect of a brand on itself. The zero *df* and missing variances and standard errors are correct since the cross-effect of an alternative on itself is zero. These results look fine. Everything that should be estimable is estimable, and everything that should not be estimable is not.

Next, we run some further checks by looking at the coded design. Before we look at the coded design, recall that the design for the first five choice sets is as follows:

```
                                  Consumer Food Product Example

                                 Client    Client                      Private
           Shelf   Client         Line     Micro/  Regional  Private   Micro/    National
   Block   Talker  Brand       Extension   Stove    Brand     Label     Stove    Competitor

     1     No      $1.29         $2.39     Micro    $1.99       N       Micro      $1.99
                   $1.29         $1.89     Stove      N         N       Stove        N
                     N           $2.39     Micro      N       $2.29     Stove        N
                   $1.29         $1.39     Stove    $2.49     $2.29     Stove        N
                   $2.09         $1.39     Micro    $1.99     $1.49     Stove        N
```

The coded design that the `%ChoicEff` macro creates is called TMP_CAND. We can look at the coded data set in several ways. First, the `Brand`, `Price`, microwave and shelf talker factors, for just the available alternatives for the first five choice sets are displayed as follows:

```
proc print data=tmp_cand(obs=24) label;
   var Brand Price Shelf Micro;
   where w;
   run;
```

The results are as follows:

```
                                  Consumer Food Product Example

                           Obs    Brand        Price    Shelf    Micro

                             1    Client       $1.29     No      Stove
                             2    Extension    $2.39     No      Micro
                             3    Regional     $1.99     No      Stove
                             5    National     $1.99     No      Stove
                             6    None         $0.00     No      Stove
                             7    Client       $1.29     No      Stove
                             8    Extension    $1.89     No      Stove
                            12    None         $0.00     No      Stove
                            14    Extension    $2.39     No      Micro
                            16    Private      $2.29     No      Stove
                            18    None         $0.00     No      Stove
                            19    Client       $1.29     No      Stove
                            20    Extension    $1.39     No      Stove
                            21    Regional     $2.49     No      Stove
                            22    Private      $2.29     No      Stove
                            24    None         $0.00     No      Stove
```

```
                  25    Client       $2.09     No      Stove
                  26    Extension    $1.39     No      Micro
                  27    Regional     $1.99     No      Stove
                  28    Private      $1.49     No      Stove
                  30    None         $0.00     No      Stove

                  34    Private      $2.29     No      Micro
                  35    National     $1.99     No      Stove
                  36    None         $0.00     No      Stove
```

Unlike all previous examples, the number of alternatives is not the same in all of the choice sets due to differing subsets of brands being unavailable in each choice set.

The coded factors for the brand effects and alternative-specific price effects for the first choice set are displayed as follows:

```
proc print data=tmp_cand(obs=5) label;
   id Brand;
   var BrandClient -- BrandNational;
   where w;
   run;

proc format; value zer 0 = '  0'; run;

proc print data=tmp_cand(obs=5) label;
   id Brand Price;
   var BrandClientPrice -- BrandNationalPrice;
   format BrandClientPrice -- BrandNationalPrice zer5.2;
   where w;
   run;
```

The results are as follows:

```
                         Consumer Food Product Example

         Brand        Client    Extension    Regional    Private    National

         Client          1          0            0           0           0
         Extension       0          1            0           0           0
         Regional        0          0            1           0           0
         National        0          0            0           0           1
         None            0          0            0           0           0
```

```
                           Consumer Food Product Example

                              Client    Extension    Regional    Private    National
      Brand         Price     Price       Price        Price      Price       Price

      Client        $1.29      1.29         0            0          0           0
      Extension     $2.39      0          2.39           0          0           0
      Regional      $1.99      0            0          1.99         0           0
      National      $1.99      0            0            0          0         1.99
      None          $0.00      0            0            0          0           0
```

The brand effects and alternative-specific price effect codings are similar to those we have used previously. The difference is the presence of all zero columns for unavailable alternatives, in this case the private label and national brands. Note that `Brand Price` are just ID variables and do not enter into the analysis.

The shelf talker and microwave coded factors (along with the `Brand`, `Price`, `Shelf`, and `Micro` factors) are displayed by the following step:

```
proc print data=tmp_cand(obs=5) label;
   id Brand Price Shelf Micro;
   var shelftalker micromicro;
   where w;
   run;
```

The results are as follows:

```
                           Consumer Food Product Example

                                                     Shelf
                Brand         Price    Shelf    Micro    Talker    Micro

                Client        $1.29     No      Stove      0         0
                Extension     $2.39     No      Micro      0         1
                Regional      $1.99     No      Stove      0         0
                National      $1.99     No      Stove      0         0
                None          $0.00     No      Stove      0         0
```

The following steps display the price cross-effects along with `Brand` and `Price` for the first choice set:

```
proc print data=tmp_cand(obs=4) label;
   id Brand Price;
   var x1Brand:; format x1Brand: zer5.2;
   where w;
   run;
```

```
proc print data=tmp_cand(obs=4) label;
   id Brand Price;
   var x2Brand:; format x2Brand: zer5.2;
   where w;
   run;

proc print data=tmp_cand(obs=4) label;
   id Brand Price;
   var x5Brand:; format x5Brand: zer5.2;
   where w;
   run;

proc print data=tmp_cand(obs=4) label;
   id Brand Price;
   var x6Brand:; format x6Brand: zer5.2;
   where w;
   run;

proc print data=tmp_cand(obs=4) label;
   id Brand Price;
   var x8Brand:; format x8Brand: zer5.2;
   where w;
   run;
```

The cross-effects are displayed in panels. This first panel shows the terms that capture the effect of the client brand on the utility of the other brands. The second panel shows the terms that capture the effect of the line extension on the other alternatives, and so on. An unavailable brand has no effect on any other brand's utility in that choice set. The results are as follows:

```
                         Consumer Food Product Example

                                     CE, Client  CE, Client              CE, Client
                         CE, Client      on          on      CE, Client      on
 Brand        Price     on Client    Extension    Regional   on Private   National

 Client       $1.29       1.29          0            0            0          0
 Extension    $2.39       0            1.29          0            0          0
 Regional     $1.99       0             0           1.29          0          0
 National     $1.99       0             0            0            0         1.29

                                         CE,         CE,                     CE,
                            CE,       Extension   Extension      CE,       Extension
                         Extension       on          on       Extension       on
 Brand        Price      on Client   Extension    Regional    on Private   National

 Client       $1.29        2.39          0           0            0           0
 Extension    $2.39        0            2.39         0            0           0
 Regional     $1.99        0             0          2.39          0           0
 National     $1.99        0             0           0            0          2.39
```

```
                                       CE,        CE,                     CE,
                          CE,       Regional    Regional       CE,      Regional
                        Regional       on          on        Regional      on
 Brand        Price    on Client   Extension    Regional    on Private   National

 Client       $1.29      1.99          0           0            0           0
 Extension    $2.39       0           1.99         0            0           0
 Regional     $1.99       0            0          1.99          0           0
 National     $1.99       0            0           0            0          1.99

                       CE,          CE,          CE,          CE,          CE,
                     Private on   Private on   Private on   Private on   Private on
 Brand       Price   Client       Extension    Regional     Private      National

 Client      $1.29     0            0            0            0            0
 Extension   $2.39     0            0            0            0            0
 Regional    $1.99     0            0            0            0            0
 National    $1.99     0            0            0            0            0

                                       CE,        CE,                     CE,
                          CE,       National    National       CE,      National
                        National       on          on        National      on
 Brand        Price    on Client   Extension    Regional    on Private   National

 Client       $1.29      1.99          0           0            0           0
 Extension    $2.39       0           1.99         0            0           0
 Regional     $1.99       0            0          1.99          0           0
 National     $1.99       0            0           0            0          1.99
```

A column like `'CE, Client on Extension'` in the first panel, for example, captures the effect of the client brand at $1.29 on the utility of the extension. In the next panel, `'CE, Extension on Client'` captures the effect of the extension at $2.39 on the utility of the client brand.

The following steps displays the availability cross-effects along with `Brand` and `Price` for the first choice set:

```
proc print data=tmp_cand(obs=4) label;
   id Brand Price;
   var a1Brand:;
   where w;
   run;

proc print data=tmp_cand(obs=4) label;
   id Brand Price;
   var a2Brand:;
   where w;
   run;
```

```
proc print data=tmp_cand(obs=4) label;
   id Brand Price;
   var a5Brand:;
   where w;
   run;

proc print data=tmp_cand(obs=4) label;
   id Brand Price;
   var a6Brand:;
   where w;
   run;

proc print data=tmp_cand(obs=4) label;
   id Brand Price;
   var a8Brand:;
   where w;
   run;
```

The availability cross-effects are displayed in panels. The first panel shows the terms that capture the effect of the client brand which on the other available alternatives, and so on. Panels with 1's in them show the effects of the available brands and panels with negative numbers show the effects of the unavailable brands. The results are as follows:

```
                        Consumer Food Product Example

                                 AE, Client  AE, Client              AE, Client
                      AE, Client     on          on      AE, Client      on
 Brand       Price    on Client  Extension    Regional   on Private   National

 Client      $1.29        0          0           0           0           0
 Extension   $2.39        0          1           0           0           0
 Regional    $1.99        0          0           1           0           0
 National    $1.99        0          0           0           0           1

                                     AE,         AE,                     AE,
                         AE,     Extension   Extension       AE,      Extension
                      Extension      on          on       Extension      on
Brand        Price    on Client  Extension    Regional   on Private   National

Client       $1.29        1          0           0            0           0
Extension    $2.39        0          0           0            0           0
Regional     $1.99        0          0           1            0           0
National     $1.99        0          0           0            0           1
```

```
                                  AE,        AE,                    AE,
                        AE,     Regional   Regional      AE,      Regional
                     Regional      on         on      Regional       on
Brand        Price   on Client  Extension  Regional   on Private  National

Client       $1.29       1          0          0          0          0
Extension    $2.39       0          1          0          0          0
Regional     $1.99       0          0          0          0          0
National     $1.99       0          0          0          0          1

                       AE,         AE,         AE,         AE,         AE,
                    Private on  Private on  Private on  Private on  Private on
Brand        Price    Client    Extension   Regional     Private    National

Client       $1.29     -2           0          0           0           0
Extension    $2.39      0          -2          0           0           0
Regional     $1.99      0           0         -2           0           0
National     $1.99      0           0          0           0          -2

                                  AE,        AE,                    AE,
                        AE,     National   National      AE,      National
                     National      on         on      National       on
Brand        Price   on Client  Extension  Regional   on Private  National

Client       $1.29       1          0          0          0          0
Extension    $2.39       0          1          0          0          0
Regional     $1.99       0          0          1          0          0
National     $1.99       0          0          0          0          0
```

The design looks good, it has reasonably good balance and correlations, it can be used to estimate all of the effects of interest, and we have shown that we know how to code all of the factors for a model with cross-effects and availability cross-effects. We are ready to collect data.

There is one more test that should be run before a design is used. The `%MktDups` macro checks the design to see if any choice sets are duplicates of any other choice sets as follows:

```
%mktdups(branded, data=sasuser.EntreeChDes,
         nalts=6, factors=brand price shelf micro)
```

The results are as follows:

```
Design:          Branded
Factors:         brand price shelf micro
                 Brand
                 Micro Price Shelf
Duplicate Sets:  0
```

The first line of the table tells us that this is a branded design as opposed to a generic design (bundles of attributes with no brands). The second line tells us the factors as specified in the `factors=` option. These are followed by the actual variable names for the factors. The last line reports the number of duplicates. In this case, there are no duplicate choice sets. If there are duplicate choice sets, then changing the random number seed might help. Changing other aspects of the design or the approach for making the design might help as well.

Generating Artificial Data

We do not illustrate questionnaire generation for this example since we have done it several times before in previous examples. Instead we go straight to data processing and analysis. The following DATA step generates some artificial data:[†]

```
%let m   = 6;
%let mm1 = %eval(&m - 1);
%let n   = 36;

proc format;
   value yn    1 = 'No'    2 = 'Talker';
   value micro 1 = 'Micro' 2 = 'Stove';
   run;
```

[†]This section shows in a cursory way how to generate artificial data. Some researchers wisely like to use artificial data to test a design before spending lots of money on collecting data. See the section beginning on page 393 for a detailed discussion of generating artificial data.

```
data _null_;
   array brands[&m] _temporary_ (5 7 1 2 3 -2);
   array u[&m];
   array x[&mm1] x1 x2 x5 x6 x8;
   do rep = 1 to 300;
      if mod(rep, 2) then put;
      put rep 3. +2 @@;
      do j = 1 to &n;
         set sasuser.EntreeLinDesLab point=j;
         do brand = 1 to &m; u[brand] = brands[brand] + 2 * normal(17); end;
         do brand = 1 to &mm1;
            if n(x[brand]) then u[brand] + -x[brand]; else u[brand] = .;
            end;
         if n(u2) and x4 = 2 then u2 + 1; /* shelf talker */
         if n(u2) and x3 = 1 then u2 + 1; /* microwave    */
         if n(u4) and x7 = 1 then u4 + 1; /* microwave    */
         * Choose the most preferred alternative.;
         m = max(of u1-u&m);
         do brand = 1 to &m;
            if n(u[brand]) then if abs(u[brand] - m) < 1e-4 then c = brand;
            end;
         put +(-1) c @@;
         end;
      end;
   stop;
   run;
```

Creating artificial data and trying the analysis before collecting real data is another way to test the design before going to the expense of data collection. The following DATA step reads the data:

```
data results;
   input Subj (choose1-choose&n) (1.) @@;
   datalines;
  1 22222415521222252222122122222121252 2   2 22222542124222212222121222222121132 2
  3 21222452154522212222122222112111252 2   4 21212512121222212222122242122121252 2
  5 22222512521222212252121222222121242 2   6 22212552311222212222422121211124252 2
  .
  .
  .
297 22222552121222242222422252212115162 2 298 22222412124212242222122251222121252 2
299 11222512121212212152122221221121262 2 300 12222512324212212222121122212321232 2
;
```

Processing the Data

The analysis proceeds in a fashion similar to before. We have already made the choice design, so we just have to merge it with the data. The data and design are merged in the usual way using the `%MktMerge` macro. Notice at this point that the unavailable alternatives are still in the design. The

`%MktMerge` macro has an `nalts=` option and expects a constant number of alternatives in each choice set. The following steps perform the merge and display a subset of the results:

```
   %mktmerge(design=sasuser.EntreeChDes, data=results, out=res2,
             nsets=&n, nalts=&m, setvars=choose1-choose&n)

   proc print data=res2(obs=12); id subj set; by subj set; run;
```

The data and design for the first two choice sets for the first subject, including the unavailable alternatives, are as follows:

```
                          Consumer Food Product Example

            B             P     M    S
        S   r             r     i    h
        u S a             i     c    e
        b e n             c     r    l      x      x      x      x      x a a  a  a  a
        j t d             e     o    f      1      2      5      6      8 1 2  5  6  8 w c

        1 1 Client    $1.29 Stove No $1.29 $2.39 $1.99 $0.00 $1.99 0 1  1 -2  1 1 2
            Extension $2.39 Micro No $1.29 $2.39 $1.99 $0.00 $1.99 1 0  1 -2  1 1 1
            Regional  $1.99 Stove No $1.29 $2.39 $1.99 $0.00 $1.99 1 1  0 -2  1 1 2
            Private   $0.00 Micro No $1.29 $2.39 $1.99 $0.00 $1.99 1 1  1  0  1 0 2
            National  $1.99 Stove No $1.29 $2.39 $1.99 $0.00 $1.99 1 1  1 -2  0 1 2
            None      $0.00 Stove No $1.29 $2.39 $1.99 $0.00 $1.99 1 1  1 -2  1 1 2
        1 2 Client    $1.29 Stove No $1.29 $1.89 $0.00 $0.00 $0.00 0 1 -2 -2 -2 1 2
            Extension $1.89 Stove No $1.29 $1.89 $0.00 $0.00 $0.00 1 0 -2 -2 -2 1 1
            Regional  $0.00 Stove No $1.29 $1.89 $0.00 $0.00 $0.00 1 1  0 -2 -2 0 2
            Private   $0.00 Stove No $1.29 $1.89 $0.00 $0.00 $0.00 1 1 -2  0 -2 0 2
            National  $0.00 Stove No $1.29 $1.89 $0.00 $0.00 $0.00 1 1 -2 -2  0 0 2
            None      $0.00 Stove No $1.29 $1.89 $0.00 $0.00 $0.00 1 1 -2 -2 -2 1 2
```

These next steps aggregate the data and display a subset of the results:

```
   proc summary data=res2 nway;
      class set brand price shelf micro x1 x2 x5 x6 x8 a1 a2 a5 a6 a8 c;
      output out=agg(drop=_type_);
      where w; /* exclude unavailable, w = 0 */
      run;

   proc print; where set = 1; run;
```

The data set is fairly large at 64,800 observations, and aggregating greatly reduces its size, which makes both the TRANSREG and the PHREG steps run in just a few seconds. This step also excludes the unavailable alternatives. When `w` is 1 (true) the alternative is available and counted, otherwise when `w` is 0 (false) the alternative is unavailable and excluded by the `where` clause and not counted. There is nothing in subsequent steps that assumes a fixed number of alternatives.

All of the variables used in the analysis are named as `class` variables in PROC SUMMARY, which reduces the data set from 64,800 observations to 286. The aggregated data for the first choice set is as follows:

```
                          Consumer Food Product Example

                                                                             _
     B           P S   M                                                     F
     r           r h   i                                                     R
O  S a           i e   c                                                     E
b  e n           c l   r      x     x     x     x     x a a a  a a           Q
s  t d           e f   o      1     2     5     6     8 1 2 5  6 8 c         _

 1 1 Client    $1.29 No Stove $1.29 $2.39 $1.99 $0.00 $1.99 0 1 1 -2 1 1  74
 2 1 Client    $1.29 No Stove $1.29 $2.39 $1.99 $0.00 $1.99 0 1 1 -2 1 2 226
 3 1 Extension $2.39 No Micro $1.29 $2.39 $1.99 $0.00 $1.99 1 0 1 -2 1 1 220
 4 1 Extension $2.39 No Micro $1.29 $2.39 $1.99 $0.00 $1.99 1 0 1 -2 1 2  80
 5 1 National  $1.99 No Stove $1.29 $2.39 $1.99 $0.00 $1.99 1 1 1 -2 0 1   6
 6 1 National  $1.99 No Stove $1.29 $2.39 $1.99 $0.00 $1.99 1 1 1 -2 0 2 294
 7 1 None      $0.00 No Stove $1.29 $2.39 $1.99 $0.00 $1.99 1 1 1 -2 1 2 300
 8 1 Regional  $1.99 No Stove $1.29 $2.39 $1.99 $0.00 $1.99 1 1 0 -2 1 2 300
```

In the first choice set, the client brand is chosen (`c` = 1) a total of `_freq_` = 74 times and not chosen (`c` = 2) a total of `_freq_` = 226 times. Each alternative is chosen and not chosen a total of 300 times, which is the number of subjects. These next steps code and run the analysis.

Cross-Effects

This next step codes the design for analysis. This coding is discussed on page 509. PROC TRANSREG is run like before, except now the data set `Agg` is specified and the ID variable includes `_freq_` (the frequency variable) but not `Subj` (the subject number variable). The following step does the coding:

```
proc transreg data=agg design=5000 nozeroconstant norestoremissing;
   model class(brand / zero='None')
         class(brand / zero='None' separators='' ' ') * identity(price)
         class(shelf micro / lprefix=5 0 zero='No' 'Stove')
         identity(x1 x2 x5 x6 x8) *
            class(brand / zero='None' separators=' ' ' on ')
         identity(a1 a2 a5 a6 a8) *
            class(brand / zero='None' separators=' ' ' on ') /
         lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   id set c _freq_;
   label shelf = 'Shelf Talker'
         micro = 'Microwave';
   run;
```

Note that like we saw in the `%ChoicEff` macro, PROC TRANSREG produces the following warning:

```
WARNING: This usage of * sets one group's slope to zero.  Specify |
         to allow all slopes and intercepts to vary.  Alternatively,
         specify CLASS(vars) * identity(vars) identity(vars) for
         separate within group functions and a common intercept.
         This is a change from Version 6.
```

This is because `class` is interacted with `identity` using the asterisk instead of the vertical bar. In a linear model, this might be a sign of a coding error, so the procedure displays a warning. If you get this warning while coding a choice model specifying `zero=`*'constant-alternative-level'*, you can safely ignore it.

The analysis is the same as we have done previously with aggregate data. PROC PHREG is run to fit the mother logit model, complete with availability cross-effects as follows:

```
proc phreg data=coded;
   strata set;
   model c*c(2) = &_trgind / ties=breslow;
   freq _freq_;
   run;
```

Multinomial Logit Model Results

Recall that we used `%phchoice(on)` on page 287 to customize the output from PROC PHREG. These steps produced the following results:

```
                         Consumer Food Product Example

                              The PHREG Procedure

                               Model Information

                      Data Set                 WORK.CODED
                      Dependent Variable       c
                      Censoring Variable       c
                      Censoring Value(s)       2
                      Frequency Variable       _FREQ_
                      Ties Handling            BRESLOW

                    Number of Observations Read         284
                    Number of Observations Used         284
                    Sum of Frequencies Read           47400
                    Sum of Frequencies Used           47400
```

```
Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                          Number of          Chosen          Not
Stratum    Set         Alternatives    Alternatives       Chosen

      1    1                   1500             300         1200
      2    2                    900             300          600
      3    3                    900             300          600
      4    4                   1500             300         1200
      5    5                   1500             300         1200
      6    6                    900             300          600
      7    7                   1500             300         1200
      8    8                   1500             300         1200
      9    9                   1500             300         1200
     10    10                  1500             300         1200
     11    11                   900             300          600
     12    12                  1500             300         1200
     13    13                  1500             300         1200
     14    14                   900             300          600
     15    15                   900             300          600
     16    16                  1500             300         1200
     17    17                  1500             300         1200
     18    18                  1500             300         1200
     19    19                  1500             300         1200
     20    20                  1200             300          900
     21    21                   900             300          600
     22    22                  1200             300          900
     23    23                  1500             300         1200
     24    24                  1500             300         1200
     25    25                  1500             300         1200
     26    26                  1500             300         1200
     27    27                  1500             300         1200
     28    28                  1500             300         1200
     29    29                  1500             300         1200
     30    30                   900             300          600
     31    31                  1500             300         1200
     32    32                  1500             300         1200
     33    33                  1200             300          900
     34    34                  1200             300          900
     35    35                  1200             300          900
     36    36                  1200             300          900
---------------------------------------------------------------
  Total                       47400           10800        36600

                         Convergence Status

           Convergence criterion (GCONV=1E-8) satisfied.
```

```
                              Model Fit Statistics

                                    Without          With
                       Criterion     Covariates     Covariates

                       -2 LOG L       154710.28      134305.63
                       AIC            154710.28      134409.63
                       SBC            154710.28      134788.57

                       Testing Global Null Hypothesis: BETA=0

               Test                 Chi-Square       DF     Pr > ChiSq

               Likelihood Ratio     20404.6461       52         <.0001
               Score                22883.7078       52         <.0001
               Wald                  6444.0844       52         <.0001

                        Multinomial Logit Parameter Estimates

                                         Parameter    Standard
                                    DF    Estimate       Error  Chi-Square  Pr > ChiSq

     Client                          1     8.16629     3.96395      4.2442      0.0394
     Extension                       1    10.30298     4.13379      6.2119      0.0127
     National                        1     5.41386     4.90468      1.2184      0.2697
     Private                         1     4.90773     4.06749      1.4558      0.2276
     Regional                        1     4.96459     5.93423      0.6999      0.4028
     Client Price                    1    -1.11653     0.77149      2.0945      0.1478
     Extension Price                 1    -0.99948     1.21987      0.6713      0.4126
     National Price                  1     1.25938     1.74132      0.5231      0.4695
     Private Price                   1    -1.33471     0.76283      3.0614      0.0802
     Regional Price                  1    -1.22852     1.48246      0.6867      0.4073
     Shelf Talker                    1     0.66941     0.07828     73.1204      <.0001
     Micro                           1     0.59645     0.06746     78.1706      <.0001
     CE, Client on Client            0           0           .           .           .
     CE, Client on Extension         1    -0.31640     0.78240      0.1635      0.6859
     CE, Client on National          1    -0.50555     0.80031      0.3990      0.5276
     CE, Client on Private           1    -0.25802     0.82061      0.0989      0.7532
     CE, Client on Regional          1     1.15121     1.03011      1.2489      0.2638
     CE, Extension on Client         1    -0.52993     1.22364      0.1876      0.6650
     CE, Extension on Extension      0           0           .           .           .
     CE, Extension on National       1    -0.55507     1.24852      0.1977      0.6566
     CE, Extension on Private        1     0.20613     1.25789      0.0269      0.8698
     CE, Extension on Regional       1    -0.54337     1.43547      0.1433      0.7050
     CE, Regional on Client          1    -1.14955     1.07675      1.1398      0.2857
     CE, Regional on Extension       1    -1.43726     1.08276      1.7620      0.1844
     CE, Regional on National        1    -1.81230     1.13204      2.5629      0.1094
     CE, Regional on Private         1    -1.20206     1.09592      1.2031      0.2727
     CE, Regional on Regional        0           0           .           .           .
```

```
CE, Private on Client        1    -0.42457     0.75836      0.3134      0.5756
CE, Private on Extension     1    -0.35800     0.75937      0.2223      0.6373
CE, Private on National      1    -0.68966     0.79742      0.7480      0.3871
CE, Private on Private       0           0           .           .           .
CE, Private on Regional      1    -1.11543     1.08771      1.0516      0.3051
CE, National on Client       1     1.42556     1.75683      0.6584      0.4171
CE, National on Extension    1     1.03538     1.75043      0.3499      0.5542
CE, National on National     0           0           .           .           .
CE, National on Private      1     1.46740     1.78874      0.6730      0.4120
CE, National on Regional     1    -0.28269     2.28193      0.0153      0.9014
AE, Client on Client         0           0           .           .           .
AE, Client on Extension      1     0.12477     0.38019      0.1077      0.7428
AE, Client on National       1     0.10606     0.38579      0.0756      0.7834
AE, Client on Private        1    -0.04026     0.39633      0.0103      0.9191
AE, Client on Regional       1    -0.57219     0.48525      1.3904      0.2383
AE, Extension on Client      1     0.77428     0.65342      1.4041      0.2360
AE, Extension on Extension   0           0           .           .           .
AE, Extension on National    1     0.61514     0.67002      0.8429      0.3586
AE, Extension on Private     1     0.18324     0.67377      0.0740      0.7856
AE, Extension on Regional    1     0.38862     0.76269      0.2596      0.6104
AE, Regional on Client       1     0.87692     0.77389      1.2840      0.2572
AE, Regional on Extension    1     1.05490     0.77497      1.8529      0.1734
AE, Regional on National     1     1.29670     0.79530      2.6584      0.1030
AE, Regional on Private      1     0.98393     0.77581      1.6085      0.2047
AE, Regional on Regional     0           0           .           .           .
AE, Private on Client        1     0.29125     0.48172      0.3655      0.5454
AE, Private on Extension     1     0.26656     0.48436      0.3029      0.5821
AE, Private on National      1     0.49015     0.50341      0.9480      0.3302
AE, Private on Private       0           0           .           .           .
AE, Private on Regional      1     0.81907     0.74339      1.2140      0.2705
AE, National on Client       1    -1.15849     1.35844      0.7273      0.3938
AE, National on Extension    1    -0.88510     1.35042      0.4296      0.5122
AE, National on National     0           0           .           .           .
AE, National on Private      1    -1.32585     1.38251      0.9197      0.3376
AE, National on Regional     1     0.31543     1.74206      0.0328      0.8563
```

Since the number of alternatives is not constant within each choice set, the summary table has non-constant numbers of alternatives and numbers of alternatives not chosen. The number chosen, 300 (or one per subject per choice set), is constant, since each subject always chooses one alternative from each choice set regardless of the number of alternatives. The total number of alternatives ranges from 900 with three alternatives to 1500 with five alternatives.

The cross-effects are mostly nonsignificant. Since most of the cross-effects are nonsignificant, we can rerun the analysis with a simpler model as follows:

```
proc transreg data=agg design=5000 nozeroconstant norestoremissing;
   model class(brand / zero='None')
         class(brand / zero='None' separators='' ' ') * identity(price)
         class(shelf micro / lprefix=5 0 zero='No' 'Stove') /
         lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   id set c _freq_;
   label shelf = 'Shelf Talker'
         micro = 'Microwave';
   run;

proc phreg data=coded;
   strata set;
   model c*c(2) = &_trgind / ties=breslow;
   freq _freq_;
   run;
```

The parameter estimates for the simpler model are as follows:

```
                        Multinomial Logit Parameter Estimates

                                 Parameter     Standard
                          DF      Estimate        Error    Chi-Square    Pr > ChiSq

     Client                1       5.65644      0.20231      781.6872        <.0001
     Extension             1       6.42043      0.21086      927.1112        <.0001
     National              1       0.77822      0.63713        1.4919        0.2219
     Private               1       4.51101      0.30491      218.8745        <.0001
     Regional              1       1.71388      0.99673        2.9567        0.0855
     Client Price          1      -0.76985      0.09446       66.4224        <.0001
     Extension Price       1      -0.50666      0.08350       36.8162        <.0001
     National Price        1       0.74444      0.29078        6.5542        0.0105
     Private Price         1      -1.33357      0.14151       88.8068        <.0001
     Regional Price        1      -0.42010      0.46037        0.8327        0.3615
     Shelf Talker          1       0.71984      0.06941      107.5588        <.0001
     Micro                 1       0.58407      0.05632      107.5642        <.0001
```

The most to least preferred brands are: client line extension, client brand, private label, the regional competitor, the national brand, and the none alternative (with an implicit part-worth utility of zero). The price effects are mostly negative, and the positive effects are only marginally significant. Both the shelf talker and the microwaveable option have positive utility.

Modeling Subject Attributes

This example uses the same design and data as we just saw, but this time we have some demographic information about our respondents that we wish to model. The following DATA step reads a subject number, the choices, and the respondent age and income (in thousands of dollars):

```
data results;
   input Subj (choose1-choose&n) (1.) age income;
   datalines;
  1 222224155212222522221221222221212522  33 109
  2 222225421242222122221212222221211322  56 117
  3 212224521545222122221222221121112522  56  78
  4 212125121212222122221222421221212522  57 107
  .
  .
  .
299 112225121212122212152122221221121262 2  41  89
300 122225123242122122221211222123212322  38  95
;
```

Merging the data and design is no different from what we saw previously. To make this analysis simpler, we do not fit any cross-effects or availability cross-effects, although we certainly could. The following step performs the merge and displays a subset of the results:

```
%mktmerge(design=sasuser.EntreeChDes(drop=x1--x8 a1--a8), data=results,
          out=res2, nsets=&n, nalts=&m, setvars=choose1-choose&n)

proc print data=res2;
   by subj set; id subj set;
   where (subj = 1 and set = 1) or
         (subj = 2 and set = 2) or
         (subj = 3 and set = 3) or
         (subj = 300 and set = 36);
   run;
```

A small sample of the data is as follows:

```
                         Consumer Food Product Example

Subj    Set    Age    Income    Brand        Price    Micro    Shelf    w    c

   1      1     33      109     Client       $1.29    Stove    No       1    2
                33      109     Extension    $2.39    Micro    No       1    1
                33      109     Regional     $1.99    Stove    No       1    2
                33      109     Private      $0.00    Micro    No       0    2
                33      109     National     $1.99    Stove    No       1    2
                33      109     None         $0.00    Stove    No       1    2
```

```
     2       2     56       117     Client       $1.29     Stove    No         1    2
                   56       117     Extension    $1.89     Stove    No         1    1
                   56       117     Regional     $0.00     Stove    No         0    2
                   56       117     Private      $0.00     Stove    No         0    2
                   56       117     National     $0.00     Stove    No         0    2
                   56       117     None         $0.00     Stove    No         1    2

     3       3     56        78     Client       $0.00     Stove    No         0    2
                   56        78     Extension    $2.39     Micro    No         1    1
                   56        78     Regional     $0.00     Stove    No         0    2
                   56        78     Private      $2.29     Stove    No         1    2
                   56        78     National     $0.00     Stove    No         0    2
                   56        78     None         $0.00     Stove    No         1    2

   300      36     38        95     Client       $2.09     Stove    No         1    2
                   38        95     Extension    $2.39     Micro    Talker     1    1
                   38        95     Regional     $0.00     Stove    No         0    2
                   38        95     Private      $0.00     Stove    No         0    2
                   38        95     National     $2.39     Stove    No         1    2
                   38        95     None         $0.00     Stove    No         1    2
```

Note that like before, the unavailable alternatives are required for the merge step. You can see that the demographic information matches the raw data and is constant within each subject. The rest of the data processing is virtually the same as well. Since we have demographic information, we do not aggregate. There would have to be ties in both the demographics and choice for aggregation to have any effect.

We use PROC TRANSREG to code, adding `Age` and `Income` to the analysis as follows:

```
proc transreg data=res2 design=5000 nozeroconstant norestoremissing;
   model class(brand / zero='None')
         identity(age income) * class(brand / zero='None' separators='' ', ')
         class(brand / zero='None' separators='' ' ') * identity(price)
         class(shelf micro / lprefix=5 0 zero='No' 'Stove') /
         lprefix=0 order=data;

   output out=code(drop=_type_ _name_ intercept);
   id subj set c w;
   label shelf = 'Shelf Talker'
         micro = 'Microwave';
   run;

data coded(drop=w); set code; where w; run; /* exclude unavailable */
```

The `Age` and `Income` variables are incorporated into the analysis by interacting them with `Brand`. Demographic variables must be interacted with product attributes to have any effect. If `identity(age income)` had been specified instead of `identity(age income) * class(brand / ...)` the coefficients for age and income would be zero. This is because age and income are constant within each choice set and subject combination, which means they are constant within each stratum. The second separator `', '` is used to create names for the brand/demographic interaction terms like `'Age, Client'`.

The following steps display the first coded choice set:

```
proc print data=coded(obs=4) label;
   id brand price;
   var BrandClient -- BrandPrivate Shelf Micro c;
   run;

proc print data=coded(obs=4 drop=Age) label;
   id brand price;
   var Age:;
   run;

proc print data=coded(obs=4 drop=Income) label;
   id brand price;
   var Income:;
   run;

proc print data=coded(obs=4) label;
   id brand price;
   var BrandClientPrice -- BrandPrivatePrice;
   format BrandClientPrice -- BrandPrivatePrice best4.;
   run;
```

The coded data set for the first subject and choice set is displayed next in a series of panels. The part that is new is the second and third panel, which are used to capture the brand by age and brand by income effects. The attributes and the brand effects are as follows:

```
                          Consumer Food Product Example

                                                               Shelf
     Brand      Price  Client  Extension  Regional  Private  Talker  Microwave  c

     Client     $1.29     1        0          0         0        No      Stove    2
     Extension  $2.39     0        1          0         0        No      Micro    1
     Regional   $1.99     0        0          1         0        No      Stove    2
     National   $1.99     0        0          0         0        No      Stove    2
```

The age by brand effects are as follows:

```
                          Consumer Food Product Example

                          Age,        Age,        Age,        Age,        Age,
     Brand        Price  Client    Extension    Regional    Private    National

     Client       $1.29    33          0           0           0           0
     Extension    $2.39     0         33           0           0           0
     Regional     $1.99     0          0          33           0           0
     National     $1.99     0          0           0           0          33
```

The income by brand effects are as follows:

```
                          Consumer Food Product Example

                        Income,     Income,      Income,     Income,     Income,
    Brand        Price   Client    Extension    Regional    Private    National

    Client       $1.29    109          0            0           0           0
    Extension    $2.39      0        109            0           0           0
    Regional     $1.99      0          0          109           0           0
    National     $1.99      0          0            0           0         109
```

The alternative-specific price effects are as follows:

```
                          Consumer Food Product Example

                             Client    Extension    Regional    Private
           Brand        Price    Price        Price       Price       Price

           Client       $1.29     1.29           0           0          0
           Extension    $2.39        0        2.39           0          0
           Regional     $1.99        0           0        1.99          0
           National     $1.99        0           0           0          0
```

The PROC PHREG specification is the same as we have used before with nonaggregated data. The following step performs the analysis:

```
proc phreg data=coded brief;
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;
```

This step took just about one minute and produced the following results:

```
                           Consumer Food Product Example

                                The PHREG Procedure

                                 Model Information

                      Data Set                    WORK.CODED
                      Dependent Variable          c
                      Censoring Variable          c
                      Censoring Value(s)          2
                      Ties Handling               BRESLOW

                    Number of Observations Read       47400
                    Number of Observations Used       47400

         Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                  Number of       Number of          Chosen          Not
        Pattern     Choices    Alternatives    Alternatives       Chosen

              1        2400               3               1            2
              2        1800               4               1            3
              3        6600               5               1            4

                                Convergence Status

                 Convergence criterion (GCONV=1E-8) satisfied.

                               Model Fit Statistics

                                       Without           With
                    Criterion       Covariates     Covariates

                    -2 LOG L         31508.579      10939.139
                    AIC              31508.579      10983.139
                    SBC              31508.579      11143.460

                    Testing Global Null Hypothesis: BETA=0

            Test                 Chi-Square       DF     Pr > ChiSq

            Likelihood Ratio     20569.4401       22         <.0001
            Score                21088.4411       22         <.0001
            Wald                  6947.1766       22         <.0001
```

```
                         Consumer Food Product Example

                              The PHREG Procedure

                     Multinomial Logit Parameter Estimates

                                 Parameter      Standard
                          DF      Estimate         Error    Chi-Square    Pr > ChiSq

    Client                 1       2.04997       0.81287        6.3599        0.0117
    Extension              1       0.47882       0.81714        0.3434        0.5579
    Regional               1      -1.49923       1.51428        0.9802        0.3221
    Private                1       3.33861       0.85485       15.2528        <.0001
    National               1      -0.62955       1.06206        0.3514        0.5533
    Age, Client            1       0.01142       0.00944        1.4655        0.2261
    Age, Extension         1       0.00855       0.00949        0.8111        0.3678
    Age, Regional          1       0.01114       0.01205        0.8548        0.3552
    Age, Private           1       0.00826       0.00970        0.7247        0.3946
    Age, National          1       0.00809       0.00964        0.7042        0.4014
    Income, Client         1       0.03227       0.00771       17.4954        <.0001
    Income, Extension      1       0.05717       0.00776       54.2991        <.0001
    Income, Regional       1       0.02496       0.01014        6.0642        0.0138
    Income, Private        1       0.00957       0.00794        1.4521        0.2282
    Income, National       1       0.00929       0.00789        1.3858        0.2391
    Client Price           1      -0.76510       0.09598       63.5410        <.0001
    Extension Price        1      -0.51364       0.08465       36.8207        <.0001
    Regional Price         1      -0.27824       0.46406        0.3595        0.5488
    Private Price          1      -1.37957       0.14286       93.2538        <.0001
    National Price         1       0.82684       0.29260        7.9852        0.0047
    Shelf Talker           1       0.74026       0.07033      110.7751        <.0001
    Micro                  1       0.59312       0.05692      108.6006        <.0001
```

In other examples, when we use the `brief` option to produce a brief summary of the strata, the table has only one line. In this case, since our choice sets have 3, 4, or 5 alternatives, we have three rows, one for each choice set size. The coefficients for the age and income variables are generally not very significant in this analysis except an effect for income on the client brand and particularly on the extension.

Allocation of Prescription Drugs

This example discusses an allocation study, which is a technique often used in the area of prescription drug marketing research. This example discusses designing the allocation experiment, processing the data, analyzing frequencies, analyzing proportions, coding, analysis, and results. The principles of designing an allocation study are the same as for designing a first-choice experiment, as is the coding and final analysis. However, processing the data before analysis is different.

The previous examples have all modeled simple choice. However, sometimes the response of interest is not simple first choice. For example, in prescription drug marketing, researchers often use allocation studies where multiple, not single choices are made. Physicians are asked questions like "For the next ten prescriptions you write for a particular condition, how many would you write for each of these drugs?" The response, for example, could be "5 for drug 1, none for drug 2, 3 for drug 3, and 2 for drug 4."

Designing the Allocation Experiment

In this study, physicians are asked to specify which of ten drugs they would prescribe to their next ten patients. In this study, ten drugs, Drug 1 — Drug 10, are available each at three different prices, \$50, \$75, and \$100. In real studies, real brand names are used and there would probably be more attributes. Since experimental design has been covered in some detail in other examples, we chose a simple design for this experiment so that we could concentrate on data processing. First, we use the `%MktRuns` autocall macro to suggest a design size. (All of the autocall macros used in this book are documented starting on page 803.) We specify `3 ** 10` for the 10 three-level factors as follows:

```
title 'Allocation of Prescription Drugs';

%mktruns(3 ** 10)
```

The results are as follows:

```
                    Allocation of Prescription Drugs

                             Design Summary

                         Number of
                         Levels       Frequency

                              3           10

                    Allocation of Prescription Drugs

          Saturated      = 21
          Full Factorial = 59,049
```

```
                  Some Reasonable                         Cannot Be
                     Design Sizes       Violations     Divided By

                               27 *              0
                               36 *              0
                               45 *              0
                               54 *              0
                               21 S             45     9
                               24               45     9
                               30               45     9
                               33               45     9
                               39               45     9
                               42               45     9

            * - 100% Efficient design can be made with the MktEx macro.
            S - Saturated Design - The smallest design that can be made.

                         Allocation of Prescription Drugs

           n    Design                                         Reference

          27             3 ** 13                             Fractional-Factorial
          36    2 ** 11  3 ** 12                             Orthogonal Array
          36    2 **  4  3 ** 13                             Orthogonal Array
          36    2 **  2  3 ** 12   6 **  1                   Orthogonal Array
          36             3 ** 13   4 **  1                   Orthogonal Array
          36             3 ** 12  12 **  1                   Orthogonal Array
          45             3 ** 10   5 **  1                   Orthogonal Array
          54    2 **  1  3 ** 25                             Orthogonal Array
          54    2 **  1  3 ** 21   9 **  1                   Orthogonal Array
          54             3 ** 24   6 **  1                   Orthogonal Array
          54             3 ** 20   6 **  1   9 **  1    Orthogonal Array
          54             3 ** 18  18 **  1                   Orthogonal Array
```

We need at least 21 choice sets and we see the optimal sizes are all divisible by nine. We use 27 choice sets, which can give us up to 13 three-level factors.

Next, we use the `%MktEx` macro to create the design.* In addition, one more factor is added to the design. This factor is used to block the design into three blocks of size 9. The following step generates the design:

```
%let nalts = 10;

%mktex(3 ** &nalts 3, n=27, seed=396)
```

*Due to machine, SAS release, and macro differences, you might not get exactly the same design used in this book, but the differences should be slight.

The macro finds a 100% *D*-efficient design. The results are as follows:

```
                          Allocation of Prescription Drugs

                              Algorithm Search History

                                    Current          Best
              Design    Row,Col  D-Efficiency  D-Efficiency  Notes
              ----------------------------------------------------------
                   1      Start      100.0000      100.0000  Tab
                   1        End      100.0000

                          Allocation of Prescription Drugs

                                The OPTEX Procedure

                              Class Level Information

                              Class  Levels    -Values-

                                x1        3    1 2 3
                                x2        3    1 2 3
                                x3        3    1 2 3
                                x4        3    1 2 3
                                x5        3    1 2 3
                                x6        3    1 2 3
                                x7        3    1 2 3
                                x8        3    1 2 3
                                x9        3    1 2 3
                                x10       3    1 2 3
                                x11       3    1 2 3

                          Allocation of Prescription Drugs

                                The OPTEX Procedure

                                                                    Average
                                                                   Prediction
       Design                                                       Standard
       Number     D-Efficiency     A-Efficiency     G-Efficiency      Error
       ------------------------------------------------------------------------
            1         100.0000         100.0000         100.0000         0.9230
```

The `%MktEx` macro always creates factor names of `x1`, `x2`, and so on with values of 1, 2, ... You can create a data set with the names and values you want and use it to rename the factors and reset the levels. This first step creates a data set with 11 variables, `Block` and `Brand1 - Brand10`. `Block` has values 1, 2, and 3, and the brand variables have values of 50, 75, and 100 with a `dollar` format. The `%MktLab` macro takes the `data=Randomized` design data set and uses the names, values, and formats in the `key=Key` data set to make the `out=Final` data set. This data set is sorted by block and displayed. The `%MktEval` macro is called to check the results. The following steps process and evaluate the design:

```
data key(drop=i);
   input Block Brand1;
   array Brand[10];
   do i = 2 to 10; brand[i] = brand1; end;
   format brand: dollar4.;
   datalines;
1  50
2  75
3 100
;

proc print; run;

%mktlab(data=randomized, key=key)


proc sort out=sasuser.DrugAlloLinDes; by block; run;

proc print; id block; by block; run;

%mkteval(blocks=block)
```

The `key=` data set is as follows:

```
                          Allocation of Prescription Drugs

Obs Block Brand1 Brand2 Brand3 Brand4 Brand5 Brand6 Brand7 Brand8 Brand9 Brand10

 1    1     $50    $50    $50    $50    $50    $50    $50    $50    $50    $50
 2    2     $75    $75    $75    $75    $75    $75    $75    $75    $75    $75
 3    3    $100   $100   $100   $100   $100   $100   $100   $100   $100   $100
```

The `%MktLab` macro displays the following mapping information:

```
Variable Mapping:
   x1  : Block
   x2  : Brand1
   x3  : Brand2
   x4  : Brand3
   x5  : Brand4
   x6  : Brand5
   x7  : Brand6
   x8  : Brand7
   x9  : Brand8
   x10 : Brand9
   x11 : Brand10
```

The design is as follows:

```
                        Allocation of Prescription Drugs

Block Brand1 Brand2 Brand3 Brand4 Brand5 Brand6 Brand7 Brand8 Brand9 Brand10

  1     $50    $75    $50    $75   $100   $100   $100   $100    $50   $100
       $100    $50   $100    $75    $75    $75   $100    $50   $100    $75
        $50    $50    $75   $100    $50    $75    $50    $75    $50    $75
        $75    $50    $50    $50   $100    $75    $75   $100    $75    $75
        $75    $75   $100   $100    $75   $100    $50    $50    $75   $100
        $50   $100   $100    $50    $75    $50    $75    $50    $50    $50
       $100    $75    $75    $50    $50   $100    $75    $75   $100   $100
       $100   $100    $50   $100   $100    $50    $50   $100   $100    $50
        $75   $100    $75    $75    $50    $50   $100    $75    $75    $50

  2    $100    $75    $50   $100    $75    $50   $100    $75    $75    $75
       $100   $100   $100    $75    $50    $75    $75   $100    $75   $100
        $50    $75   $100    $50    $50    $50    $50   $100   $100    $75
        $75    $50   $100   $100    $50   $100   $100   $100    $50    $50
        $50   $100    $75   $100   $100    $75   $100    $50   $100   $100
       $100    $50    $75    $50   $100   $100    $50    $50    $75    $50
        $50    $50    $50    $75    $75   $100    $75    $75   $100    $50
        $75    $75    $75    $75   $100    $50    $75    $50    $50    $75
        $75   $100    $50    $50    $75    $75    $50    $75    $50   $100
  3     $100    $75   $100    $75   $100    $75    $50    $75    $50    $50
         $75    $75    $50    $50    $50    $75   $100    $50   $100    $50
         $50    $75    $75   $100    $75    $75    $75   $100    $75    $50
         $50   $100    $50    $75    $50   $100    $50    $50    $75    $75
         $50    $50   $100    $50   $100    $50   $100    $75    $75   $100
         $75    $50    $75    $75    $75    $50    $50   $100   $100   $100
         $75   $100   $100   $100   $100   $100    $75    $75   $100    $75
        $100    $50    $50   $100    $50    $50    $75    $50    $50   $100
        $100   $100    $75    $50    $75   $100   $100   $100    $50    $75
```

Some of the evaluation results are as follows:

```
                          Allocation of Prescription Drugs
                       Canonical Correlations Between the Factors
                  There are 0 Canonical Correlations Greater Than 0.316

            Block Brand1 Brand2 Brand3 Brand4 Brand5 Brand6 Brand7 Brand8 Brand9 Brand10

    Block    1       0      0      0      0      0      0      0      0      0      0
    Brand1   0       1      0      0      0      0      0      0      0      0      0
    Brand2   0       0      1      0      0      0      0      0      0      0      0
    Brand3   0       0      0      1      0      0      0      0      0      0      0
    Brand4   0       0      0      0      1      0      0      0      0      0      0
    Brand5   0       0      0      0      0      1      0      0      0      0      0
    Brand6   0       0      0      0      0      0      1      0      0      0      0
    Brand7   0       0      0      0      0      0      0      1      0      0      0
    Brand8   0       0      0      0      0      0      0      0      1      0      0
    Brand9   0       0      0      0      0      0      0      0      0      1      0
    Brand10  0       0      0      0      0      0      0      0      0      0      1

                      Allocation of Prescription Drugs
                           Summary of Frequencies
              There are 0 Canonical Correlations Greater Than 0.316

                                 Frequencies

                   Block             9 9 9
                   Brand1            9 9 9
                   Brand2            9 9 9
                   Brand3            9 9 9
                   Brand4            9 9 9
                   Brand5            9 9 9
                   Brand6            9 9 9
                   Brand7            9 9 9
                   Brand8            9 9 9
                   Brand9            9 9 9
                   Brand10           9 9 9
```

```
                    Block Brand1      3 3 3 3 3 3 3 3 3
                    Block Brand2      3 3 3 3 3 3 3 3 3
                    Block Brand3      3 3 3 3 3 3 3 3 3
                    Block Brand4      3 3 3 3 3 3 3 3 3
                    Block Brand5      3 3 3 3 3 3 3 3 3
                    Block Brand6      3 3 3 3 3 3 3 3 3
                    Block Brand7      3 3 3 3 3 3 3 3 3
                    Block Brand8      3 3 3 3 3 3 3 3 3
                    Block Brand9      3 3 3 3 3 3 3 3 3
                    Block Brand10     3 3 3 3 3 3 3 3 3
                    .
                    .
                    .
                    N-Way             1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                                      1 1 1 1 1 1 1 1
```

We can also create a choice design from the linear arrangement and evaluate it using the `%ChoicEff` macro. The design is rolled into a choice design as follows using the same technique that we used in previous examples:

```
   %mktkey(Brand1-Brand10)

   data key(keep=Brand Price);
      input Brand $ 1-8 Price $;
      datalines;
   Brand  1    Brand1
   Brand  2    Brand2
   Brand  3    Brand3
   Brand  4    Brand4
   Brand  5    Brand5
   Brand  6    Brand6
   Brand  7    Brand7
   Brand  8    Brand8
   Brand  9    Brand9
   Brand 10    Brand10
   -           .
   ;

   %mktroll(design=sasuser.DrugAlloLinDes, key=key, alt=brand, out=rolled,
            options=nowarn)
```

At this point, the choice design can be stored in a permanent SAS data set stored for analysis time as we have done in previous examples. Alternatively, you an just store the linear arrangement and use the same (or similar) code to make the choice design at analysis time. We use the latter approach here. This is because a form that is convenient for analysis is not always convenient for design evaluation. This is explained in more detail once the evaluation results are displayed. For now, simply note that the constant alternative is denoted by a dash here, but we use a blank (or character missing value) for the analysis.

The choice design is evaluated as follows:

```
%choiceff(data=rolled,             /* candidate set of choice sets        */
          init=rolled(keep=set),   /* select these sets                   */
          intiter=0,               /* evaluate without internal iterations */
                                   /* model with stdz orthogonal coding   */
                                   /* ref level for brand is '-'          */
          model=class(brand price / zero='-' sta),
          options=relative,        /* display relative D-efficiency       */
          nsets=27,                /* number of choice sets               */
          nalts=11,                /* number of alternatives              */
          beta=zero)               /* assumed beta vector, Ho: b=0        */
```

Some of the results are as follows:

```
                       Allocation of Prescription Drugs

                                Final Results

                          Design                 1
                          Choice Sets           27
                          Alternatives          11
                          Parameters            12
                          Maximum Parameters   270
                          D-Efficiency     26.1557
                          Relative D-Eff   96.8729
                          D-Error           0.0382
                          1 / Choice Sets   0.0370

                       Allocation of Prescription Drugs

                                                                  Standard
 n    Variable Name    Label               Variance    DF         Error

 1    BrandBrand__1    Brand Brand  1      0.037037     1        0.19245
 2    BrandBrand__2    Brand Brand  2      0.037037     1        0.19245
 3    BrandBrand__3    Brand Brand  3      0.037037     1        0.19245
 4    BrandBrand__4    Brand Brand  4      0.037037     1        0.19245
 5    BrandBrand__5    Brand Brand  5      0.037037     1        0.19245
 6    BrandBrand__6    Brand Brand  6      0.037037     1        0.19245
 7    BrandBrand__7    Brand Brand  7      0.037037     1        0.19245
 8    BrandBrand__8    Brand Brand  8      0.037037     1        0.19245
 9    BrandBrand__9    Brand Brand  9      0.037037     1        0.19245
10    BrandBrand_10    Brand Brand 10      0.037037     1        0.19245
11    Price50          Price  50           0.044815     1        0.21170
12    Price75          Price  75           0.044815     1        0.21170
                                                       ==
                                                       12
```

Note that there are 12 parameters, one of each of the 10 brands (10 brands minus plus one constant, minus 1) and two for price (three brands minus 1). The constant brand (not shown, but represented by '-') was explicitly designated as the reference level for brand. The standardized orthogonal contrast coding is used for both attributes. Note that each of the 11 brands (including the constant) appears in each of the 11 alternatives in each choice set, the variances for `Brand` are at the minimum value of one over the number of choice sets. The price attribute does not have a perfect relationship like this, 3 prices does not evenly divide 11 alternatives, and there is a constant alternative, so the variances for the price attribute are higher. Still, they look pretty good. Relative D-efficiency is 96.8729, which is pretty good. Note that like before, relative D-efficiency = 100 is not possible since there is a constant alternative and 3 does not divide 11. So again, this design looks pretty good.

Processing the Data

Questionnaires are generated and data collected using a minor modification of the methods discussed in earlier examples. The difference is instead of asking for first choice data, allocation data are collected instead. Each row of the input data set contains a block, subject, and set number, followed by the number of times each of the ten alternatives is chosen. If all of the choice frequencies are zero, then the constant alternative is chosen. The `if` statement is used to check data entry. For convenience, choice set number is recoded to run from 1 to 27 instead of consisting of three blocks of nine sets. This gives us one fewer variable on which to stratify.

```
data results;
   input Block Subject Set @9 (freq1-freq&nalts) (2.);
   if not (sum(of freq:) in (0, &nalts)) then put _all_;
   set = (block - 1) * 9 + set;
   datalines;
1   1 1  0 0 8 0 2 0 0 0 0 0
1   1 2  0 0 8 0 0 0 2 0 0 0
1   1 3  0 0 0 0 0 0 0 010 0
1   1 4  1 0 0 1 3 3 0 0 2 0
1   1 5  2 0 8 0 0 0 0 0 0 0
1   1 6  0 1 3 1 0 0 0 0 1 4
1   1 7  0 1 3 1 1 2 0 0 2 0
1   1 8  0 0 3 0 0 2 1 0 0 4
1   1 9  0 2 5 0 0 0 0 0 3 0
2   210  1 1 0 2 0 3 0 1 1 1
2   211  1 0 3 1 0 1 1 0 2 1
.
.
.
;
```

In the first step, in creating an analysis data set for an allocation study, we reformat the data from one row per choice set per block per subject ($9 \times 3 \times 100 = 2700$ observations) to one per alternative (including the constant) per choice set per block per subject ($(10+1) \times 9 \times 3 \times 100 = 29,700$ observations).

For each choice set, 11 observations are written storing the choice frequency in the variable `Count` and the brand in the variable `Brand`. If no alternative is chosen, then the constant alternative is chosen ten times, otherwise it is chosen zero times. The following steps process and display some of the data:

```
data allocs(keep=block set brand count);
   set results;
   array freq[&nalts];

   * Handle the &nalts alternatives;
   do b = 1 to &nalts;
      Brand = 'Brand ' || put(b, 2.);
      Count = freq[b];
      output;
      end;


   * Constant alt choice is implied if nothing else is chosen.
     brand = ' ' is used to flag the constant alternative.;
   brand = ' ';
   count = 10 * (sum(of freq:) = 0);
   output;
   run;

proc print data=results(obs=3) label noobs; run;
proc print data=allocs(obs=33); id block set; by block set; run;
```

The PROC PRINT steps show how the first three observations of the `Results` data set are transposed into the first 33 observations of the `Allocs` data set. The results are as follows:

```
                        Allocation of Prescription Drugs

  Block Subject Set Freq1 Freq2 Freq3 Freq4 Freq5 Freq6 Freq7 Freq8 Freq9 Freq10

    1      1     1    0     0     8     0     2     0     0     0      0     0
    1      1     2    0     0     8     0     0     0     2     0      0     0
    1      1     3    0     0     0     0     0     0     0     0     10     0
```

```
                    Allocation of Prescription Drugs

                   Block    Set     Brand       Count

                     1       1     Brand  1        0
                                   Brand  2        0
                                   Brand  3        8
                                   Brand  4        0
                                   Brand  5        2
                                   Brand  6        0
                                   Brand  7        0
                                   Brand  8        0
                                   Brand  9        0
                                   Brand 10        0
                                                   0


                     1       2     Brand  1        0
                                   Brand  2        0
                                   Brand  3        8
                                   Brand  4        0
                                   Brand  5        0
                                   Brand  6        0
                                   Brand  7        2
                                   Brand  8        0
                                   Brand  9        0
                                   Brand 10        0
                                                   0

                     1       3     Brand  1        0
                                   Brand  2        0
                                   Brand  3        0
                                   Brand  4        0
                                   Brand  5        0
                                   Brand  6        0
                                   Brand  7        0
                                   Brand  8        0
                                   Brand  9       10
                                   Brand 10        0
                                                   0
```

The following step aggregates the data:

```
* Aggregate, store the results back in count.;

proc summary data=allocs nway missing;
   class set brand;
   output sum(count)=Count out=allocs(drop=_type_ _freq_);
   run;
```

It stores in the variable Count the number of times each alternative of each choice set is chosen. This creates a data set with 297 observations (3 blocks $\times$ 9 sets $\times$ 11 alternatives = 297).

These next steps prepare the design for analysis. We need to create a data set Key that describes how the factors in our design are used for analysis. It contains all of the factor names, Brand1, Brand2, ..., Brand10. We can run the %MktKey macro to get these names for cutting and pasting into the program without typing them as follows:

```
%mktkey(Brand1-Brand10)
```

The %MktKey macro produced the following line:

```
Brand1 Brand2 Brand3 Brand4 Brand5 Brand6 Brand7 Brand8 Brand9 Brand10
```

The next step rolls out the experimental design data set to match the choice allocations data set. The data set is transposed from one row per choice set to one row per alternative per choice set. This data set also has 297 observations. As we saw in previous examples, the %MktRoll macro can be used to process the design as follows:

```
data key(keep=Brand Price);
   input Brand $ 1-8 Price $;
   datalines;
Brand  1    Brand1
Brand  2    Brand2
Brand  3    Brand3
Brand  4    Brand4
Brand  5    Brand5
Brand  6    Brand6
Brand  7    Brand7
Brand  8    Brand8
Brand  9    Brand9
Brand 10    Brand10
.           .
;

%mktroll(design=sasuser.DrugAlloLinDes, key=key, alt=brand, out=rolled,
         options=nowarn)

proc print data=rolled(obs=11); format price dollar4.; run;
```

The results are as follows:

```
                   Allocation of Prescription Drugs

                   Obs    Set     Brand      Price

                     1     1     Brand  1      $50
                     2     1     Brand  2      $75
                     3     1     Brand  3      $50
                     4     1     Brand  4      $75
                     5     1     Brand  5     $100
                     6     1     Brand  6     $100
                     7     1     Brand  7     $100
                     8     1     Brand  8     $100
                     9     1     Brand  9      $50
                    10     1     Brand 10     $100
                    11     1                     .
```

Both data sets must be sorted the same way before they can be merged. The constant alternative, indicated by a missing brand, is last in the design choice set and hence is out of order. Missing must come before nonmissing for the merge. The order is correct in the `Allocs` data set since it is created by PROC SUMMARY with `Brand` as a `class` variable. The following step sorts the data:

```
proc sort data=rolled; by set brand; run;
```

The data are merged along with error checking to ensure that the merge proceeded properly. Both data sets should have the same observations and `Set` and `Brand` variables, so the merge should be one to one. The following steps merge the data and display a subset of the results:

```
data allocs2;
   merge allocs(in=flag1) rolled(in=flag2);
   by set brand;
   if flag1 ne flag2 then put 'ERROR: Merge is not 1 to 1.';
   format price dollar4.;
   run;

proc print data=allocs2(obs=22);
   var brand price count;
   sum count;
   by notsorted set;
   id set;
   run;
```

In the aggregate and combined data set, we see how often each alternative is chosen for each choice set. For example, in the first choice set, the constant alternative is chosen zero times, Brand 1 at $100 was chosen 103 times, and so on. The 11 alternatives are chosen a total of 1000 times, 100 subjects times 10 choices each. The results are as follows:

```
                 Allocation of Prescription Drugs

               Set     Brand       Price    Count

                1                      .        0
                       Brand  1      $50      103
                       Brand  2      $75       58
                       Brand  3      $50      318
                       Brand  4      $75       99
                       Brand  5     $100       54
                       Brand  6     $100       83
                       Brand  7     $100       71
                       Brand  8     $100       58
                       Brand  9      $50      100
                       Brand 10     $100       56
               ---                          -----
                1                            1000

                2                      .       10
                       Brand  1     $100       73
                       Brand  2      $50       76
                       Brand  3     $100      342
                       Brand  4      $75       55
                       Brand  5      $75       50
                       Brand  6      $75       77
                       Brand  7     $100       95
                       Brand  8      $50       71
                       Brand  9     $100       72
                       Brand 10      $75       79
               ---                          -----
                2                            1000
```

At this point, the data set contains 297 observations (27 choice sets times 11 alternatives) showing the number of times each alternative is chosen. This data set must be augmented to also include the number of times each alternative is not chosen. For example, in the first choice set, brand 1 is chosen 103 times, which means it is not chosen 0 + 58 + 318 + 99 + 54 + 83 + 71 + 58 + 100 + 56 = 897 times. We use a macro, `%MktAllo` for "marketing allocation study" to process the data. We specify the input `data=allocs2` data set, the output `out=allocs3` data set, the number of alternatives including the constant (`nalts=%eval(&nalts + 1)`), the variables in the data set except the frequency variable (`vars=set brand price`), and the frequency variable (`freq=Count`). The macro counts how many times each alternative is chosen and not chosen and writes the results to the `out=` data set along with the usual `c` = 1 for chosen and `c` = 2 for unchosen.

The following step processes the data and displays a subset of the results:

```
%mktallo(data=allocs2, out=allocs3, nalts=%eval(&nalts + 1),
         vars=set brand price, freq=Count)

proc print data=allocs3(obs=22);
   var set brand price count c;
   run;
```

The first 22 records of the allocation data set are as follows:

```
                 Allocation of Prescription Drugs

 Obs    Set     Brand       Price    Count    c

   1     1                    .          0    1
   2     1                    .       1000    2
   3     1     Brand  1     $50        103    1
   4     1     Brand  1     $50        897    2
   5     1     Brand  2     $75         58    1
   6     1     Brand  2     $75        942    2
   7     1     Brand  3     $50        318    1
   8     1     Brand  3     $50        682    2
   9     1     Brand  4     $75         99    1
  10     1     Brand  4     $75        901    2
  11     1     Brand  5    $100         54    1
  12     1     Brand  5    $100        946    2
  13     1     Brand  6    $100         83    1
  14     1     Brand  6    $100        917    2
  15     1     Brand  7    $100         71    1
  16     1     Brand  7    $100        929    2
  17     1     Brand  8    $100         58    1
  18     1     Brand  8    $100        942    2
  19     1     Brand  9     $50        100    1
  20     1     Brand  9     $50        900    2
  21     1     Brand 10    $100         56    1
  22     1     Brand 10    $100        944    2
```

In the first choice set, the constant alternative is chosen zero times and not chosen 1000 times, Brand 1 is chosen 103 times and not chosen $1000 - 103 = 897$ times, Brand 2 is chosen 58 times and not chosen $1000 - 58 = 942$ times, and so on. Note that allocation studies do not always have fixed sums, so it is important to use the `%MktAllo` macro or some other approach that actually counts the number of times each alternative is not chosen. It is not always sufficient to simply subtract from a fixed constant (in this case, 1000).

Coding and Analysis

The next step codes the design for analysis. Indicator variables are created for `Brand` and `Price`. All of the PROC TRANSREG options have been discussed in other examples. The following step does the coding:

```
proc transreg design data=allocs3 nozeroconstant norestoremissing;
   model class(brand price / zero=none) / lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   id set c count;
   run;
```

Analysis proceeds like it has in all other examples. We stratify by choice set number. We do not need to stratify by `Block` since choice set number does not repeat within block. The following step performs the analysis:

```
proc phreg data=coded;
   where count > 0;
   model c*c(2) = &_trgind / ties=breslow;
   freq count;
   strata set;
   run;
```

We used the `where` statement to exclude observations with zero frequency; otherwise PROC PHREG complains about them.

Multinomial Logit Model Results

Recall that we used `%phchoice(on)` on page 287 to customize the output from PROC PHREG. The results are as follows:

```
                        Allocation of Prescription Drugs

                              The PHREG Procedure

                               Model Information

                     Data Set                    WORK.CODED
                     Dependent Variable          c
                     Censoring Variable          c
                     Censoring Value(s)          2
                     Frequency Variable          Count
                     Ties Handling               BRESLOW

                   Number of Observations Read         583
                   Number of Observations Used         583
                   Sum of Frequencies Read          297000
                   Sum of Frequencies Used          297000
```

```
Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                          Number of        Chosen          Not
Stratum    Set         Alternatives    Alternatives      Chosen

      1    1                  11000            1000       10000
      2    2                  11000            1000       10000
      3    3                  11000            1000       10000
      4    4                  11000            1000       10000
      5    5                  11000            1000       10000
      6    6                  11000            1000       10000
      7    7                  11000            1000       10000
      8    8                  11000            1000       10000
      9    9                  11000            1000       10000
     10    10                 11000            1000       10000
     11    11                 11000            1000       10000
     12    12                 11000            1000       10000
     13    13                 11000            1000       10000
     14    14                 11000            1000       10000
     15    15                 11000            1000       10000
     16    16                 11000            1000       10000
     17    17                 11000            1000       10000
     18    18                 11000            1000       10000
     19    19                 11000            1000       10000
     20    20                 11000            1000       10000
     21    21                 11000            1000       10000
     22    22                 11000            1000       10000
     23    23                 11000            1000       10000
     24    24                 11000            1000       10000
     25    25                 11000            1000       10000
     26    26                 11000            1000       10000
     27    27                 11000            1000       10000
---------------------------------------------------------------
  Total                      297000           27000      270000

                       Convergence Status

         Convergence criterion (GCONV=1E-8) satisfied.

                     Model Fit Statistics

                           Without           With
            Criterion    Covariates     Covariates

            -2 LOG L      502505.13      489062.66
            AIC           502505.13      489086.66
            SBC           502505.13      489185.11
```

```
                    Testing Global Null Hypothesis: BETA=0

            Test                Chi-Square       DF     Pr > ChiSq

            Likelihood Ratio    13442.4676       12         <.0001
            Score               18340.8415       12         <.0001
            Wald                14087.6778       12         <.0001

                     Multinomial Logit Parameter Estimates

                          Parameter      Standard
                 DF        Estimate         Error    Chi-Square    Pr > ChiSq

     Brand  1     1         2.09906       0.06766      962.5297        <.0001
     Brand  2     1         2.09118       0.06769      954.5113        <.0001
     Brand  3     1         3.54204       0.06484     2984.4698        <.0001
     Brand  4     1         2.09710       0.06766      960.5277        <.0001
     Brand  5     1         2.08523       0.06771      948.4791        <.0001
     Brand  6     1         2.03530       0.06790      898.6218        <.0001
     Brand  7     1         2.06920       0.06777      932.3154        <.0001
     Brand  8     1         2.08573       0.06771      948.9824        <.0001
     Brand  9     1         2.11705       0.06759      980.9640        <.0001
     Brand 10     1         2.06363       0.06779      926.7331        <.0001
      $50         1         0.00529       0.01628        0.1058        0.7450
      $75         1       0.0005304       0.01629        0.0011        0.9740
     $100         0               0             .             .             .
```

The output shows that there are 27 strata, one per choice set, each consisting of 1000 chosen alternatives (10 choices by 100 subjects) and 10,000 unchosen alternatives. All of the brand coefficients are "significant," with the Brand 3 effect being by far the strongest. (We will soon see that statistical significance should be ignored with allocation studies.) There is no price effect.

Analyzing Proportions

Recall that we collected data by asking physicians to report which brands they would prescribe the next ten times they write prescriptions. Alternatively, we could ask them to report the *proportion* of time they would prescribe each brand. We can simulate having proportion data by dividing our count data by 10. This means our frequency variable no longer contains integers, so we need to specify the `notruncate` option in the PROC PHREG `freq` statement to permit "noninteger frequencies." This is illustrated in the following steps:

```
data coded2;
   set coded;
   count = count / 10;
   run;
```

```
proc phreg data=coded2;
   where count > 0;
   model c*c(2) = &_trgind / ties=breslow;
   freq count / notruncate;
   strata set;
   run;
```

When we do this, we see the number of alternatives and the number chosen and not chosen decrease by a factor of 10 as do all of the Chi-Square tests. The coefficients are unchanged. This implies that market share calculations are invariant to the different scalings of the frequencies. However, the *p*-values are not invariant. The sample size is artificially inflated when counts are used so *p*-values are not interpretable in an allocation study. When proportions are used, each subject is contributing 1 to the number chosen instead of 10, just like a normal choice study, so *p*-values have meaning. The results are as follows:

```
                          Allocation of Prescription Drugs

                               The PHREG Procedure

                                Model Information

                        Data Set                 WORK.CODED2
                        Dependent Variable       c
                        Censoring Variable       c
                        Censoring Value(s)       2
                        Frequency Variable       Count
                        Ties Handling            BRESLOW

                      Number of Observations Read         583
                      Number of Observations Used         583
                      Sum of Frequencies Read           29700
                      Sum of Frequencies Used           29700

           Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                                 Number of          Chosen           Not
           Stratum    Set     Alternatives    Alternatives        Chosen

                 1    1             1100.0           100.0        1000.0
                 2    2             1100.0           100.0        1000.0
                 3    3             1100.0           100.0        1000.0
                 4    4             1100.0           100.0        1000.0
                 5    5             1100.0           100.0        1000.0
                 6    6             1100.0           100.0        1000.0
                 7    7             1100.0           100.0        1000.0
                 8    8             1100.0           100.0        1000.0
                 9    9             1100.0           100.0        1000.0
                10    10            1100.0           100.0        1000.0
```

```
      11    11                1100.0            100.0      1000.0
      12    12                1100.0            100.0      1000.0
      13    13                1100.0            100.0      1000.0
      14    14                1100.0            100.0      1000.0
      15    15                1100.0            100.0      1000.0
      16    16                1100.0            100.0      1000.0
      17    17                1100.0            100.0      1000.0
      18    18                1100.0            100.0      1000.0
      19    19                1100.0            100.0      1000.0
      20    20                1100.0            100.0      1000.0
      21    21                1100.0            100.0      1000.0
      22    22                1100.0            100.0      1000.0
      23    23                1100.0            100.0      1000.0
      24    24                1100.0            100.0      1000.0
      25    25                1100.0            100.0      1000.0
      26    26                1100.0            100.0      1000.0
      27    27                1100.0            100.0      1000.0
---------------------------------------------------------------
  Total                      29700.0           2700.0     27000.0

                          Convergence Status

          Convergence criterion (GCONV=1E-8) satisfied.

                        Model Fit Statistics

                             Without          With
             Criterion     Covariates    Covariates

             -2 LOG L       37816.553     36472.307
             AIC            37816.553     36496.307
             SBC            37816.553     36567.119

             Testing Global Null Hypothesis: BETA=0

     Test                 Chi-Square       DF     Pr > ChiSq

     Likelihood Ratio      1344.2468       12         <.0001
     Score                 1834.0841       12         <.0001
     Wald                  1408.7678       12         <.0001
```

```
                                Multinomial Logit Parameter Estimates

                           Parameter      Standard
                   DF       Estimate         Error    Chi-Square    Pr > ChiSq

        Brand  1    1        2.09906       0.21395       96.2530        <.0001
        Brand  2    1        2.09118       0.21404       95.4511        <.0001
        Brand  3    1        3.54204       0.20503      298.4470        <.0001
        Brand  4    1        2.09710       0.21398       96.0528        <.0001
        Brand  5    1        2.08523       0.21411       94.8479        <.0001
        Brand  6    1        2.03530       0.21470       89.8622        <.0001
        Brand  7    1        2.06920       0.21430       93.2315        <.0001
        Brand  8    1        2.08573       0.21411       94.8982        <.0001
        Brand  9    1        2.11705       0.21375       98.0964        <.0001
        Brand 10    1        2.06363       0.21436       92.6733        <.0001
         $50        1        0.00529       0.05148        0.0106        0.9181
         $75        1      0.0005304       0.05152        0.0001        0.9918
        $100        0              0             .         .             .
```

Chair Design with Generic Attributes

This study illustrates creating an experimental design for a purely generic choice model. This example discusses generic attributes, alternative swapping, choice set swapping, and constant alternatives. In a purely generic study, there are no brands, just bundles of attributes. Also see page 102 in the experimental design chapter for examples of how to combinatorially construct optimal generic choice designs for certain problems. You can find additional examples in the documentation for the `%ChoicEff` macro beginning on page 806.

Say a manufacturer is interested in designing one or more new chairs. The manufacturer can vary the attributes of the chairs, present subjects with competing chair designs, and model the effects of the attributes on choice. The attributes of interest are as follows:

Factor	Attribute	Levels
X1	Color	3 Colors
X2	Back	3 Styles
X3	Seat	3 Styles
X4	Arm Rest	3 Styles
X5	Material	3 Materials

Since seeing descriptions of chairs is not the same as seeing and sitting in the actual chairs, the manufacturer is going to actually make sample chairs for people to try and choose from. Subjects are shown groups of three chairs at a time. If we were to make our design using the approach discussed in previous examples, we would use the `%MktEx` autocall macro to create a design with 15 factors, five for the first chair, five for the second chair, and five for the third chair. This design would need at least $15 \times (3 - 1) + 1 = 31$ runs and 93 sample chairs. The following step shows how we could have made the design:[*]

```
title 'Generic Chair Attributes';

* This design will not be used;
%mktex(3 ** 15, n=36, seed=238)

%mktkey(3 5)

%mktroll(design=randomized, key=key, out=cand)
```

The `%MktEx` approach to designing an experiment like this allows you to fit very general models including models with alternative-specific effects and even mother logit models. However, at analysis time for this purely generic model, we fit a model with 10 parameters, two for each of the five factors, `class(x1-x5)`. Creating a design with over $31 \times 3 = 93$ chairs is way too expensive. In ordinary linear model designs, we need at least as many runs as parameters. In choice designs, we need to count the total number of alternatives across all choice sets, subtract the number of choice sets, and this number must be at least as large as the number of parameters. Equivalently, each choice set allows us to estimate $m - 1$ parameters, where m is the number of alternatives in that choice set. In this case, we could fit our purely generic model with as few as $10/(3 - 1) = 5$ choice sets.

[*]Due to machine, SAS release, and macro differences, you might not get exactly the same design used in this book, but the differences should be slight.

Since we only need a simple generic model for this example, and since our chair manufacturing for our research is expensive, we do not use the `%MktEx` approach for designing our choice experiment. Instead, we use a different approach that lets us get a smaller design that is adequate for our model and budget. Recall the discussion of linear model design efficiency, choice model design efficiency, and using linear model design efficiency as a surrogate for choice design goodness starting on page 62. Instead of using linear model design efficiency as a surrogate for choice design goodness, we can directly optimize choice design efficiency given an assumed model and parameter vector $\boldsymbol{\beta}$. This approach uses the `%ChoicEff` macro.

Generic Attributes, Alternative Swapping, Large Candidate Set

This part of the example illustrates using the `%ChoicEff` macro for efficient choice designs, using its algorithm that builds a design from candidate alternatives (as opposed to candidates consisting of entire choice sets). First, we use the `%MktRuns` macro to suggest a candidate-set size as follows:

```
   %mktruns(3 ** 5)
```

Some of the results are as follows:

```
                    Generic Chair Attributes

                         Design Summary

                    Number of
                    Levels       Frequency

                        3            5

      Saturated      = 11
      Full Factorial = 243

      Some Reasonable                      Cannot Be
         Design Sizes       Violations     Divided By

                 18 *                0
                 27 *                0
                 36 *                0
                 12                 10     9
                 15                 10     9
                 21                 10     9
                 24                 10     9
                 30                 10     9
                 33                 10     9
                 11 S               15     3 9
```

```
* - 100% Efficient design can be made with the MktEx macro.
S - Saturated Design - The smallest design that can be made.
    Note that the saturated design is not one of the
    recommended designs for this problem.  It is shown
    to provide some context for the recommended sizes.


                    Generic Chair Attributes

  n    Design                                 Reference

 18    2 **  1  3 **  7                      Orthogonal Array
 18             3 **  6  6 **  1             Orthogonal Array
 27             3 ** 13                      Fractional-Factorial
 27             3 **  9  9 **  1             Fractional-Factorial
 36    2 ** 11  3 ** 12                      Orthogonal Array
 36    2 ** 10  3 **  8  6 **  1             Orthogonal Array
 36    2 **  4  3 ** 13                      Orthogonal Array
 36    2 **  3  3 **  9  6 **  1             Orthogonal Array
 36    2 **  2  3 ** 12  6 **  1             Orthogonal Array
 36    2 **  2  3 **  5  6 **  2             Orthogonal Array
 36    2 **  1  3 **  8  6 **  2             Orthogonal Array
 36             3 ** 13  4 **  1             Orthogonal Array
 36             3 ** 12 12 **  1             Orthogonal Array
 36             3 **  7  6 **  3             Orthogonal Array
```

We could use candidate sets of size: 18, 27 or 36. Additionally, since this problem is small, we could try an 81-run fractional-factorial design or the 243-run full-factorial design. We choose the 243-run full-factorial design, since it is reasonably small and it gives the macro the most freedom to find a good design.[*]

We use the `%MktEx` macro to create a candidate set. The candidate set consists of 5 three-level factors, one for each of the five generic attributes. The following steps create the candidates:

```
%mktex(3 ** 5, n=243)

proc print data=design(obs=27); run;
```

Part of the candidate set is as follows:

[*]Later, we will see we could have chosen 18.

```
                    Generic Chair Attributes

                 Obs    x1    x2    x3    x4    x5

                   1     1     1     1     1     1
                   2     1     1     1     1     2
                   3     1     1     1     1     3
                   4     1     1     1     2     1
                   5     1     1     1     2     2
                   6     1     1     1     2     3
                   7     1     1     1     3     1
                   8     1     1     1     3     2
                   9     1     1     1     3     3
                  10     1     1     2     1     1
                  11     1     1     2     1     2
                  12     1     1     2     1     3
                  13     1     1     2     2     1
                  14     1     1     2     2     2
                  15     1     1     2     2     3
                  16     1     1     2     3     1
                  17     1     1     2     3     2
                  18     1     1     2     3     3
                  19     1     1     3     1     1
                  20     1     1     3     1     2
                  21     1     1     3     1     3
                  22     1     1     3     2     1
                  23     1     1     3     2     2
                  24     1     1     3     2     3
                  25     1     1     3     3     1
                  26     1     1     3     3     2
                  27     1     1     3     3     3
```

Next, we search that candidate set for an efficient design for the model specification `class(x1-x5)` and the assumption $\boldsymbol{\beta} = \mathbf{0}$. We use the `%ChoicEff` autocall macro to do this. (All of the autocall macros used in this book are documented starting on page 803.) This approach is based on the work of Huber and Zwerina (1996) who proposed constructing efficient experimental designs for choice experiments under an assumed model and $\boldsymbol{\beta}$. The `%ChoicEff` macro uses a modified Fedorov algorithm (Fedorov 1972; Cook and Nachtsheim 1980) to optimize the choice model variance matrix. We are using the largest possible candidate set for this problem, the full-factorial design, and we ask for more than the default number of iterations, so run time is slower than it could be. However, we are requesting a very small number of choice sets. Building the chairs is expensive, so we want to get a really good but small design. The following specification requests a generic design with six choice sets each consisting of three alternatives:

```
%choiceff(data=design,              /* candidate set of alternatives        */
          model=class(x1-x5 / sta), /* model with stdzd orthogonal coding   */
          nsets=6,                  /* number of choice sets                */
          maxiter=100,              /* maximum number of designs to make    */
          seed=121,                 /* random number seed                   */
          flags=3,                  /* 3 alternatives, generic candidates   */
          options=relative,         /* display relative D-efficiency        */
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

The `data=final` option names the input data set of candidates. The `model=class(x1-x5 / sta)` option specifies the most general model that is considered at analysis time. This is a main-effects model for `x1-x5` with a standardized orthogonal contrast coding. You must specify this coding if you want to see relative *D*-efficiency on a 0 to 100 scale. The `nsets=6` option specifies the number of choice sets. Note that this is considerably smaller than the minimum of 31 that is required if we are just using the `%MktEx` linear-arrangement approach ($6 \times 3 = 18$ chairs instead of $31 \times 3 = 93$ chairs). The `maxiter=100` option requests 100 designs based on 100 random initial designs (by default, `maxiter=2`). The `seed=121` option specifies the random number seed. The `flags=3` option specifies that there are three alternatives in a purely generic design.[†] The `options=relative` option scales *D*-efficiency relative to the hypothetical maximum value so that we can get a 0 to 100 scale for *D*-efficiency. Relative *D*-efficiency is not displayed by default since it is only meaningful for some designs. The `beta=zero` option specifies the assumption $\boldsymbol{\beta} = \mathbf{0}$. A vector of numbers like `beta=-1 0 -1 0 -1 0 -1 0 -1 0 -1 0` could be specified. (See page 878 for an example of this.) When you wish to assume all parameters are zero, you can specify `beta=zero` instead of typing a vector of the zeros. You can also omit the `beta=` option if you just want the macro to list the parameters. You can use this list to ensure that you specify the parameters in the right order.

The first part of the output from the macro is a list of all of the effects generated and the assumed values of $\boldsymbol{\beta}$. The effects are as follows:

```
                         Generic Chair Attributes

                          n    Name    Beta    Label

                          1    x11       0     x1 1
                          2    x12       0     x1 2
                          3    x21       0     x2 1
                          4    x22       0     x2 2
                          5    x31       0     x3 1
                          6    x32       0     x3 2
                          7    x41       0     x4 1
                          8    x42       0     x4 2
                          9    x51       0     x5 1
                         10    x52       0     x5 2
```

[†]The option name `flags=` comes from the fact that in the context of other types of designs (designs with brands or labeled alternatives), this option provides a set of "flag" variables that specify which candidates can be be used for which alternatives.

It is very important that you check this list and make sure it is correct. In particular, when you are explicitly specifying the $\boldsymbol{\beta}$ vector, you need to make sure you specified all of the values in the right order. If you are not specifying a beta vector you can check the names and labels in the final table of variances and standard errors.

Next, the macro produces the iteration history, which is different from the iteration histories we are used to seeing in the `%MktEx` macro. The `%ChoicEff` macro uses PROC IML and a modified Fedorov algorithm to iteratively improve the efficiency of the choice design given the specified candidates, model, and $\boldsymbol{\beta}$. Note that these unscaled efficiencies are *not* on a 0 to 100 scale. This step took about 2 minutes. Some of the results are as follows:

```
                         Generic Chair Attributes

               Design   Iteration  D-Efficiency      D-Error
               ---------------------------------------------
                    1       0          1.83063 *     0.54626
                            1          4.91557 *     0.20344
                            2          5.20220 *     0.19223
                            3          5.40987 *     0.18485
                            4          5.42657 *     0.18428
                 .
                 .
                 .

               Design   Iteration  D-Efficiency      D-Error
               ---------------------------------------------
                   34       0          2.44100       0.40967
                            1          4.77565       0.20940
                            2          5.49875       0.18186
                            3          6.00000 *     0.16667
                            4          6.00000       0.16667
                 .
                 .
                 .

               Design   Iteration  D-Efficiency      D-Error
               ---------------------------------------------
                  100       0          2.37105       0.42175
                            1          5.22899       0.19124
                            2          5.41804       0.18457
                            3          5.41804       0.18457
```

An asterisk is used to indicate places where D-efficiency is greater than any previously reported value.

Next, the macro shows which design it chose and the final D-efficiency and D-error (D-efficiency = 1 / D-error). The results are as follows:

```
                          Generic Chair Attributes

                               Final Results

                         Design                 34
                         Choice Sets             6
                         Alternatives            3
                         Parameters             10
                         Maximum Parameters     12
                         D-Efficiency       6.0000
                         Relative D-Eff   100.0000
                         D-Error            0.1667
                         1 / Choice Sets    0.1667
```

Next, it shows the variance, standard error, and *df* for each effect as follows:

```
                          Generic Chair Attributes

                Variable                                    Standard
         n        Name      Label     Variance    DF         Error

         1        x11       x1 1       0.16667     1       0.40825
         2        x12       x1 2       0.16667     1       0.40825
         3        x21       x2 1       0.16667     1       0.40825
         4        x22       x2 2       0.16667     1       0.40825
         5        x31       x3 1       0.16667     1       0.40825
         6        x32       x3 2       0.16667     1       0.40825
         7        x41       x4 1       0.16667     1       0.40825
         8        x42       x4 2       0.16667     1       0.40825
         9        x51       x5 1       0.16667     1       0.40825
        10        x52       x5 2       0.16667     1       0.40825
                                                  ==
                                                  10
```

It is important to ensure that each effect is estimable: $(df = 1)$. Usually, when all of the variances are constant, like we see in this table, it means that the macro has found the optimal design. In fact, in this case, we see that all of the variances in the last table equal both *D*-Error and one over the number of choice sets in the preceding table. This means that relative *D* efficiency, which is also shown in the preceding table must equal 100. With the standardized orthogonal contrast coding, and a situation like this where an optimal design is possible, *D*-efficiency ranges from 0 to the number of choice sets. In the optimal design, *D*-efficiency = 6, the number of sets, and relative *D*-efficiency is 100 times *D*-efficiency divided by the number of choice sets.

The data set Best contains the final, best design found. It is displayed as follows:

```
proc print; by set; id set; run;
```

The data set contains: Design – the number of the design with the maximum *D*-efficiency,
Efficiency – the *D*-efficiency of this design,
Index – the candidate set observation number,
Set – the choice set number,
Prob – the probability that this alternative will be chosen given $\boldsymbol{\beta}$,
n – the observation number,
x1-x5 – the design, and
_f1-_f3 – the automatically generated alternative flags.
The data set is as follows:

```
                              Generic Chair Attributes

  Set  Design  Efficiency  Index    Prob     n   x1  x2  x3  x4  x5  _f1  _f2  _f3

   1     34        6        183   0.33333  595   3   1   3   1   3    1    1    1
         34        6         62   0.33333  596   1   3   1   3   2    1    1    1
         34        6        121   0.33333  597   2   2   2   2   1    1    1    1

   2     34        6        217   0.33333  598   3   3   1   1   1    1    1    1
         34        6         45   0.33333  599   1   2   2   3   3    1    1    1
         34        6        104   0.33333  600   2   1   3   2   2    1    1    1

   3     34        6        215   0.33333  601   3   2   3   3   2    1    1    1
         34        6        147   0.33333  602   2   3   2   1   3    1    1    1
         34        6          4   0.33333  603   1   1   1   2   1    1    1    1

   4     34        6         78   0.33333  604   1   3   3   2   3    1    1    1
         34        6        178   0.33333  605   3   1   2   3   1    1    1    1
         34        6        110   0.33333  606   2   2   1   1   2    1    1    1

   5     34        6         90   0.33333  607   2   1   1   3   3    1    1    1
         34        6         46   0.33333  608   1   2   3   1   1    1    1    1
         34        6        230   0.33333  609   3   3   2   2   2    1    1    1

   6     34        6        195   0.33333  610   3   2   1   2   3    1    1    1
         34        6         11   0.33333  611   1   1   2   1   2    1    1    1
         34        6        160   0.33333  612   2   3   3   3   1    1    1    1
```

This design has 18 runs (6 choice sets × 3 alternatives). Notice that in this design, each level occurs exactly once in each factor and each choice set. To use this design for analysis, you only need the variables Set and x1-x5. Since it is already in choice design format, it does not need to be processed using the %MktRoll macro. Since data collection, processing, and analysis have already been covered in detail in other examples, this example concentrates solely on experimental design.

There is one more test that should be run before a design is used. The `%MktDups` macro checks the design to see if any choice sets are duplicates of any other choice sets. The following steps evaluate the design:

```
%mktdups(generic, data=best, nalts=3, factors=x1-x5)
```

The results are as follows:

```
Design:          Generic
Factors:         x1-x5
                 x1 x2 x3 x4 x5
Sets w Dup Alts: 0
Duplicate Sets:  0
```

The first line of the table tells us that this is a generic design. The second line tells us the factors as specified in the `factors=` option. These are followed by the actual variable names for the factors. The next line reports the number of choice sets that have duplicate alternatives—that is, the number of times an alternative appears in a set more than once. The last line reports the number of duplicate choice sets. In this case, there are no duplicates of either type. If there are duplicates, then changing the random number seed might help. Changing other aspects of the design or the approach for making the design might help as well.

Generic Attributes, Alternative Swapping, Small Candidate Set

In this part of this example, we try to make an equivalent design to the one we just made, only this time using a smaller candidate set. The following steps create and evaluate the design:

```
%mktex(3 ** 5, n=18)

%choiceff(data=design,              /* candidate set of alternatives       */
          model=class(x1-x5 / sta), /* model with stdzd orthogonal coding  */
          nsets=6,                  /* number of choice sets               */
          maxiter=20,               /* maximum number of designs to make   */
          seed=121,                 /* random number seed                  */
          flags=3,                  /* 3 alternatives, generic candidates  */
          options=relative,         /* display relative D-efficiency       */
          beta=zero)                /* assumed beta vector, Ho: b=0        */

proc print; by set; id set; run;

%mktdups(generic, data=best, nalts=3, factors=x1-x5)
```

This time, instead of creating a full-factorial candidate set, we asked for 5 three-level factors from the L_{18}, an orthogonal array in 18 runs. We also asked for fewer iterations in the `%ChoicEff` macro. Since the candidate set is much smaller, the macro should be able to find the best design available in this candidate set fairly easily. Some of the results are as follows:

```
                   Generic Chair Attributes

                    n    Name    Beta    Label

                    1    x11       0     x1 1
                    2    x12       0     x1 2
                    3    x21       0     x2 1
                    4    x22       0     x2 2
                    5    x31       0     x3 1
                    6    x32       0     x3 2
                    7    x41       0     x4 1
                    8    x42       0     x4 2
                    9    x51       0     x5 1
                   10    x52       0     x5 2

                   Generic Chair Attributes

          Design   Iteration  D-Efficiency       D-Error
          ----------------------------------------------
               1       0                0              .
                       1          4.35495 *      0.22962
                       2          4.35495        0.22962

            .
            .
            .

          Design   Iteration  D-Efficiency       D-Error
          ----------------------------------------------
              20       0          1.89505        0.52769
                       1          4.44909        0.22476
                       2          5.05750        0.19773
                       3          5.37575        0.18602
                       4          6.00000        0.16667
                       5          6.00000        0.16667
```

```
                          Generic Chair Attributes

                               Final Results

                         Design                  4
                         Choice Sets             6
                         Alternatives            3
                         Parameters             10
                         Maximum Parameters     12
                         D-Efficiency       6.0000
                         Relative D-Eff   100.0000
                         D-Error            0.1667
                         1 / Choice Sets    0.1667

                          Generic Chair Attributes

                Variable                                    Standard
           n      Name      Label    Variance    DF          Error

           1      x11       x1 1      0.16667     1        0.40825
           2      x12       x1 2      0.16667     1        0.40825
           3      x21       x2 1      0.16667     1        0.40825
           4      x22       x2 2      0.16667     1        0.40825
           5      x31       x3 1      0.16667     1        0.40825
           6      x32       x3 2      0.16667     1        0.40825
           7      x41       x4 1      0.16667     1        0.40825
           8      x42       x4 2      0.16667     1        0.40825
           9      x51       x5 1      0.16667     1        0.40825
          10      x52       x5 2      0.16667     1        0.40825
                                                 ==
                                                 10

                          Generic Chair Attributes

Set  Design  Efficiency  Index    Prob    n  f1  f2  f3  x1  x2  x3  x4  x5

 1      4         6        18  0.33333  55   1   1   1   3   3   3   3   3
        4         6         1  0.33333  56   1   1   1   1   1   1   1   1
        4         6         9  0.33333  57   1   1   1   2   2   2   2   2

 2      4         6         3  0.33333  58   1   1   1   1   2   1   3   3
        4         6        12  0.33333  59   1   1   1   2   3   2   1   1
        4         6        14  0.33333  60   1   1   1   3   1   3   2   2

 3      4         6        15  0.33333  61   1   1   1   3   2   1   2   1
        4         6         8  0.33333  62   1   1   1   2   1   3   1   3
        4         6         5  0.33333  63   1   1   1   1   3   2   3   2

 4      4         6        16  0.33333  64   1   1   1   3   2   2   1   3
        4         6         6  0.33333  65   1   1   1   1   3   3   2   1
        4         6         7  0.33333  66   1   1   1   2   1   1   3   2
```

```
   5      4         6          4   0.33333  67   1   1   1   1   2   3   1   2
          4         6         13   0.33333  68   1   1   1   3   1   2   3   1
          4         6         11   0.33333  69   1   1   1   2   3   1   2   3

   6      4         6         17   0.33333  70   1   1   1   3   3   1   1   2
          4         6          2   0.33333  71   1   1   1   1   1   2   2   3
          4         6         10   0.33333  72   1   1   1   2   2   3   3   1

Design:           Generic
Factors:          x1-x5
                  x1 x2 x3 x4 x5
Sets w Dup Alts: 0
Duplicate Sets:  0
```

Notice that we got the same *D*-efficiency and variances as before (*D*-efficiency = 6 and all variances are $1/6 \approx 0.16667$). Also notice the `Index` variable in the design (which is the candidate set row number). Each candidate appears in the design exactly once, and the `%MktDups` macro confirms that there are no duplicates. As is shown in the experimental design chapter starting on page 102, for problems like this (all generic attributes, no brands, no constant alternative, total number of alternatives equal to the number of runs in an orthogonal design, all factors available in that orthogonal design, and an assumed $\boldsymbol{\beta}$ vector of zero) that the optimal design can be created by optimally sorting the rows of an orthogonal design into choice sets, and the `%ChoicEff` macro can do this quite well. More directly, this design could be made from the orthogonal array $3^6 6^1$ in 18 runs by using the six-level factor as the choice set number.

Six choice sets is a bit small. If you can afford a larger number, it would be good to try a larger design. In this case, nine choice sets are requested using a fractional-factorial candidate set in 27 runs. Notice that like before, the number of runs in the candidate set is chosen to be the product of the number of choice sets and the number of alternatives in each choice set. The following steps generate and evaluate the design:

```
%mktex(3 ** 5, n=27, seed=382)

%choiceff(data=design,              /* candidate set of alternatives       */
          model=class(x1-x5 / sta), /* model with stdzd orthogonal coding  */
          nsets=9,                  /* number of choice sets               */
          maxiter=20,               /* maximum number of designs to make   */
          seed=121,                 /* random number seed                  */
          flags=3,                  /* 3 alternatives, generic candidates  */
          options=relative,         /* display relative D-efficiency       */
          beta=zero)                /* assumed beta vector, Ho: b=0        */

proc print; id set; by set; var index prob x:; run;

%mktdups(generic, data=best, nalts=3, factors=x1-x5)
```

The results summary, effects, and the variances are as follows:

```
                         Generic Chair Attributes

                              Final Results

                        Design                  4
                        Choice Sets             9
                        Alternatives            3
                        Parameters             10
                        Maximum Parameters     18
                        D-Efficiency       9.0000
                        Relative D-Eff   100.0000
                        D-Error            0.1111
                        1 / Choice Sets    0.1111

                         Generic Chair Attributes

              Variable                                      Standard
        n       Name       Label     Variance    DF           Error

        1       x11        x1 1       0.11111     1         0.33333
        2       x12        x1 2       0.11111     1         0.33333
        3       x21        x2 1       0.11111     1         0.33333
        4       x22        x2 2       0.11111     1         0.33333
        5       x31        x3 1       0.11111     1         0.33333
        6       x32        x3 2       0.11111     1         0.33333
        7       x41        x4 1       0.11111     1         0.33333
        8       x42        x4 2       0.11111     1         0.33333
        9       x51        x5 1       0.11111     1         0.33333
       10       x52        x5 2       0.11111     1         0.33333
                                                 ==
                                                 10

                         Generic Chair Attributes

      Set    Index        Prob     x1     x2     x3     x4     x5

        1       22     0.33333      3      2      1      1      3
                 8     0.33333      1      3      2      2      1
                12     0.33333      2      1      3      3      2

        2       23     0.33333      3      2      1      2      1
                10     0.33333      2      1      3      1      3
                 9     0.33333      1      3      2      3      2

        3        6     0.33333      1      2      3      3      1
                20     0.33333      3      1      2      2      3
                16     0.33333      2      3      1      1      2
```

```
                 4        13      0.33333      2      2      2      1      1
                          26      0.33333      3      3      3      2      2
                           3      0.33333      1      1      1      3      3

                 5        24      0.33333      3      2      1      3      2
                           7      0.33333      1      3      2      1      3
                          11      0.33333      2      1      3      2      1

                 6        14      0.33333      2      2      2      2      2
                          27      0.33333      3      3      3      3      3
                           1      0.33333      1      1      1      1      1

                 7        15      0.33333      2      2      2      3      3
                          25      0.33333      3      3      3      1      1
                           2      0.33333      1      1      1      2      2

                 8        21      0.33333      3      1      2      3      1
                          17      0.33333      2      3      1      2      3
                           4      0.33333      1      2      3      1      2

                 9         5      0.33333      1      2      3      2      3
                          18      0.33333      2      3      1      3      1
                          19      0.33333      3      1      2      1      2

    Design:          Generic
    Factors:         x1-x5
                     x1 x2 x3 x4 x5
    Sets w Dup Alts: 0
    Duplicate Sets:  0
```

Notice that like before, the variances are constant, but in this case smaller at 1/9, and each candidate appears once. This is an optimal design in 9 choice sets. More directly, this design could be made from the orthogonal array $3^9 9^1$ in 27 runs by using the nine-level factor as the choice set number.

Generic Attributes, a Constant Alternative, and Alternative Swapping

Now let's make a design for the same problem but this time with a constant alternative. We first use the `%MktEx` macro just like before to make a design for the nonconstant alternatives, then we use a DATA step to add the flags and a constant alternative as follows:

```
title 'Generic Chair Attributes';

%mktex(3 ** 5, n=243, seed=306)

data final(drop=i);
   set design end=eof;
   retain f1-f3 1 f4 0;
   output;
   if eof then do;
      array x[9] x1-x5 f1-f4;
      do i = 1 to 9; x[i] = i le 5 or i eq 9; end;
      output;
      end;
   run;

proc print data=final(where=(x1 eq x3 and x2 eq x4 and x3 eq x5 or f4)); run;
```

A sample of the observations in the candidate set is as follows:

```
                         Generic Chair Attributes

          Obs    x1    x2    x3    x4    x5    f1    f2    f3    f4

            1     1     1     1     1     1     1     1     1     0
           31     1     2     1     2     1     1     1     1     0
           61     1     3     1     3     1     1     1     1     0
           92     2     1     2     1     2     1     1     1     0
          122     2     2     2     2     2     1     1     1     0
          152     2     3     2     3     2     1     1     1     0
          183     3     1     3     1     3     1     1     1     0
          213     3     2     3     2     3     1     1     1     0
          243     3     3     3     3     3     1     1     1     0
          244     1     1     1     1     1     0     0     0     1
```

The first 243 observations can be used for any of the first three alternatives (`f1 = f2 = f3 = 1, f4 = 0`) and the 244*th* observation can only be used for fourth or constant alternative (`f1 = f2 = f3 = 0, f4 = 1`). In this example, the constant alternative is composed solely from the first level of each factor. Of course this could be changed depending on the situation. The `%ChoicEff` macro invocation is the same as before, except now we have four flag variables. The following steps generate and evaluate the design:

```
%choiceff(data=final,              /* candidate set of alternatives        */
          model=class(x1-x5 / sta), /* model with stdzd orthogonal coding   */
          nsets=6,                 /* number of choice sets                */
          maxiter=100,             /* maximum number of designs to make    */
          seed=121,                /* random number seed                   */
          flags=f1-f4,             /* flag which alt can go where, 4 alts  */
          options=relative,        /* display relative D-efficiency        */
          beta=zero)               /* assumed beta vector, Ho: b=0         */

proc print; by set; id set; run;

%mktdups(generic, data=best, nalts=4, factors=x1-x5)
```

You can see in the final design that there are now four alternatives and the last alternative in each choice set is constant and is always flagged by `f4=1`. In the interest of space, most of the iteration histories are omitted. Some of the results are as follows:

```
                         Generic Chair Attributes

                          n    Name    Beta    Label

                          1    x11       0     x1 1
                          2    x12       0     x1 2
                          3    x21       0     x2 1
                          4    x22       0     x2 2
                          5    x31       0     x3 1
                          6    x32       0     x3 2
                          7    x41       0     x4 1
                          8    x42       0     x4 2
                          9    x51       0     x5 1
                         10    x52       0     x5 2

                         Generic Chair Attributes

              Design   Iteration  D-Efficiency       D-Error
              ----------------------------------------------
                   1       0           2.20693 *     0.45312
                           1           4.67998 *     0.21368
                           2           4.87965 *     0.20493
                           3           4.90282 *     0.20396

                .
                .
                .
```

```
Design   Iteration  D-Efficiency       D-Error
----------------------------------------------
    13       0          2.56694       0.38957
             1          4.54049       0.22024
             2          4.75518       0.21030
             3          4.99035       0.20039
             4          5.19495 *     0.19249
             5          5.21381 *     0.19180
   .
   .
   .

Design   Iteration  D-Efficiency       D-Error
----------------------------------------------
   100       0          2.74564       0.36421
             1          4.59264       0.21774
             2          4.80304       0.20820
             3          4.88340       0.20478

                 Generic Chair Attributes

                      Final Results

          Design                   13
          Choice Sets               6
          Alternatives              4
          Parameters               10
          Maximum Parameters       18
          D-Efficiency         5.2138
          Relative D-Eff      86.8968
          D-Error              0.1918
          1 / Choice Sets      0.1667
```

```
                          Generic Chair Attributes

                Variable                                      Standard
           n      Name       Label     Variance     DF         Error

           1      x11        x1 1       0.19116      1        0.43722
           2      x12        x1 2       0.21187      1        0.46029
           3      x21        x2 1       0.19116      1        0.43722
           4      x22        x2 2       0.21187      1        0.46029
           5      x31        x3 1       0.19965      1        0.44683
           6      x32        x3 2       0.20072      1        0.44801
           7      x41        x4 1       0.19965      1        0.44683
           8      x42        x4 2       0.20072      1        0.44801
           9      x51        x5 1       0.18850      1        0.43417
          10      x52        x5 2       0.21187      1        0.46029
                                                    ==
                                                    10

                          Generic Chair Attributes

Set  Design  Efficiency  Index  Prob   n   x1  x2  x3  x4  x5  f1  f2  f3  f4

 1     13      5.21381    152   0.25  289   2   3   2   3   2   1   1   1   0
       13      5.21381    213   0.25  290   3   2   3   2   3   1   1   1   0
       13      5.21381     15   0.25  291   1   1   2   2   3   1   1   1   0
       13      5.21381    244   0.25  292   1   1   1   1   1   0   0   0   1

 2     13      5.21381    154   0.25  293   2   3   3   1   1   1   1   1   0
       13      5.21381     15   0.25  294   1   1   2   2   3   1   1   1   0
       13      5.21381    197   0.25  295   3   2   1   3   2   1   1   1   0
       13      5.21381    244   0.25  296   1   1   1   1   1   0   0   0   1

 3     13      5.21381    108   0.25  297   2   1   3   3   3   1   1   1   0
       13      5.21381    220   0.25  298   3   3   1   2   1   1   1   1   0
       13      5.21381     38   0.25  299   1   2   2   1   2   1   1   1   0
       13      5.21381    244   0.25  300   1   1   1   1   1   0   0   0   1

 4     13      5.21381    121   0.25  301   2   2   2   2   1   1   1   1   0
       13      5.21381    182   0.25  302   3   1   3   1   2   1   1   1   0
       13      5.21381     63   0.25  303   1   3   1   3   3   1   1   1   0
       13      5.21381    244   0.25  304   1   1   1   1   1   0   0   0   1

 5     13      5.21381    111   0.25  305   2   2   1   1   3   1   1   1   0
       13      5.21381     77   0.25  306   1   3   3   2   2   1   1   1   0
       13      5.21381    178   0.25  307   3   1   2   3   1   1   1   1   0
       13      5.21381    244   0.25  308   1   1   1   1   1   0   0   0   1

 6     13      5.21381    228   0.25  309   3   3   2   1   3   1   1   1   0
       13      5.21381     52   0.25  310   1   2   3   3   1   1   1   1   0
       13      5.21381     86   0.25  311   2   1   1   2   2   1   1   1   0
       13      5.21381    244   0.25  312   1   1   1   1   1   0   0   0   1
```

```
Design:          Generic
Factors:         x1-x5
                 x1 x2 x3 x4 x5
Sets w Dup Alts: 0
Duplicate Sets:  0
```

When there are three alternatives, each alternative has a probability of choice of 1/3, and now with four alternatives, the probability is 1/4. They are all equal because of the assumption $\boldsymbol{\beta} = \mathbf{0}$. With other assumptions about $\boldsymbol{\beta}$, typically the probabilities will not all be equal. Relative *D*-efficiency is less than 100 since there is a constant alternative. It must also be less than 100 since we have all three-level factors but four alternatives. The optimal *D*-efficiency is not the number of choice sets in designs like this, so relative *D*-efficiency is less than 100.

To use this design for analysis, you only need the variables `Set` and `x1-x5`. Since it is already in choice design format (one row per alternative), it does not need to be processed using the `%MktRoll` macro. Note that when you make designs with the `%ChoicEff` macro, the `model` statement in PROC TRANSREG should match or be no more complicated than the `model` specification that generated the design:

```
model class(x1-x5);
```

A model with fewer degrees of freedom is safe, although the design is suboptimal. For example, if `x1-x5` are quantitative attributes, the following model is safe:

```
model identity(x1-x5);
```

However, specifying interactions, or using this design in a branded study and specifying alternative-specific effects like this could lead to quite a few inestimable parameters. The following statements illustrate:

```
* Bad idea for this design!!;
model class(x1-x5 x1*x2 x4*x5);

* Another bad idea for this design!!;
model class(brand)
      class(brand * x1 brand * x2 brand * x3 brand * x4 brand * x5);
```

Generic Attributes, a Constant Alternative, and Choice Set Swapping

The `%ChoicEff` macro can be used in a very different way. Instead of providing a candidate set of alternatives to swap in and out of the design, you can provide a candidate set of entire choice sets. For this particular example, swapping alternatives is almost certainly better (see page 579). However, sometimes, if you need to impose restrictions on which alternative can appear with which other alternative, then you must use the set-swapping options. We start by using the `%MktEx` macro to make a candidate design, with one run per choice set and one factor for each attribute of each alternative (just like we did in the vacation, fabric softener, and food examples). We then process the candidates from one row per choice set to one row per alternative per choice set using the `%MktRoll` macro. The following steps make the candidate set of choice sets:

```
%mktex(3 ** 15, n=81 * 81, seed=522)

%mktkey(3 5)

data key;
   input (x1-x5) ($);
   datalines;
x1     x2     x3     x4     x5
x6     x7     x8     x9     x10
x11    x12    x13    x14    x15
.      .      .      .      .
;

%mktroll(design=randomized, key=key, out=rolled)

* Code the constant alternative;
data final;
   set rolled;
   if _alt_ = '4' then do; x1 = 1; x2 = 1; x3 = 1; x4 = 1; x5 = 1; end;
   run;

proc print; by set; id set; where set in (1, 100, 1000, 5000, 6561); run;
```

The %MktKey macro produced the following data set, which we copied, pasted, and augmented to make the Key data set:

```
x1     x2     x3     x4     x5

x1     x2     x3     x4     x5
x6     x7     x8     x9     x10
x11    x12    x13    x14    x15
```

A few of the candidate choice sets are as follows:

```
              Generic Chair Attributes

Set    _Alt_    x1    x2    x3    x4    x5

  1      1       2     1     1     2     3
         2       3     3     1     1     1
         3       2     2     1     1     3
         4       1     1     1     1     1
```

```
                        100     1      1     2     3     1     2     1
                                2      2     3     3     2     1
                                3      2     2     3     3     2
                                4      1     1     1     1     1

                       1000     1      2     1     2     1     3
                                2      2     1     2     1     2
                                3      1     3     2     2     2
                                4      1     1     1     1     1

                       5000     1      3     1     3     2     3
                                2      3     3     3     3     2
                                3      1     2     1     2     3
                                4      1     1     1     1     1

                       6561     1      1     3     1     2     2
                                2      3     2     2     2     2
                                3      1     3     3     1     3
                                4      1     1     1     1     1
```

Next, we run the `%ChoicEff` macro, only this time we specify `nalts=4` instead of `flags=f1-f4`. Since there are no alternative flag variables to count, we have to tell the macro how many alternatives are in each choice set. The option `nalts=` also specifies that the candidate set consists of candidate choice sets, whereas `flags=` specifies that the candidate set consists of candidate alternatives. We ask for fewer iterations since the candidate set is large. The following steps generate and evaluate the design:

```
   %choiceff(data=final,              /* candidate set of choice sets         */
            model=class(x1-x5 / sta), /* model with stdzd orthogonal coding   */
            nsets=6,                  /* number of choice sets                */
            nalts=4,                  /* number of alternatives               */
            maxiter=10,               /* maximum number of designs to make    */
            seed=109,                 /* random number seed                   */
            options=relative,         /* display relative D-efficiency        */
            beta=zero)                /* assumed beta vector, Ho: b=0         */

   %mktdups(generic, data=best, nalts=4, factors=x1-x5)
```

The results are as follows:

```
                              Generic Chair Attributes

                             n    Name    Beta    Label

                             1    x11       0     x1 1
                             2    x12       0     x1 2
                             3    x21       0     x2 1
```

```
                   4    x22       0     x2 2
                   5    x31       0     x3 1
                   6    x32       0     x3 2
                   7    x41       0     x4 1
                   8    x42       0     x4 2
                   9    x51       0     x5 1
                  10    x52       0     x5 2

                    Generic Chair Attributes

          Design   Iteration  D-Efficiency      D-Error
          ----------------------------------------------
               1       0          2.78600 *     0.35894
                       1          4.40738 *     0.22689
                       2          4.53259 *     0.22062
                       3          4.53259       0.22062
             .
             .
             .

          Design   Iteration  D-Efficiency      D-Error
          ----------------------------------------------
               5       0          2.75184       0.36339
                       1          4.34618       0.23009
                       2          4.47415       0.22351
                       3          4.67101 *     0.21409
                       4          4.69946 *     0.21279
                       5          4.69946       0.21279
             .
             .
             .

          Design   Iteration  D-Efficiency      D-Error
          ----------------------------------------------
              10       0          2.80475       0.35654
                       1          4.26387       0.23453
                       2          4.40048       0.22725
                       3          4.51659       0.22141
                       4          4.51659       0.22141
```

```
                              Generic Chair Attributes

                                   Final Results

                             Design                   5
                             Choice Sets              6
                             Alternatives             4
                             Parameters              10
                             Maximum Parameters      18
                             D-Efficiency        4.6995
                             Relative D-Eff     78.3243
                             D-Error             0.2128
                             1 / Choice Sets     0.1667

                              Generic Chair Attributes

                    Variable                                     Standard
               n      Name      Label    Variance    DF       Error

               1      x11       x1 1      0.19102     1      0.43705
               2      x12       x1 2      0.37907     1      0.61568
               3      x21       x2 1      0.24790     1      0.49790
               4      x22       x2 2      0.27501     1      0.52441
               5      x31       x3 1      0.19389     1      0.44033
               6      x32       x3 2      0.23730     1      0.48713
               7      x41       x4 1      0.22452     1      0.47384
               8      x42       x4 2      0.21910     1      0.46809
               9      x51       x5 1      0.21234     1      0.46081
              10      x52       x5 2      0.24140     1      0.49133
                                                     ==
                                                     10

   Design:          Generic
   Factors:         x1-x5
                    x1 x2 x3 x4 x5
   Sets w Dup Alts: 0
   Duplicate Sets:  0
```

This design is less *D*-efficient than we found using the alternative-swapping algorithm, so we will not use it.

Design Algorithm Comparisons

It is instructive to compare the three approaches outlined in this chapter in the context of this problem. There are $3^{3\times5} = 14,348,907$ choice sets for this problem (three-level factors and 3 alternatives times 5 factors per alternative). If we were to use the linear arrangement approach using the `%MktEx` macro, we could never begin to consider all possible candidate choice sets. Similarly, with the choice-set-swapping algorithm of the `%ChoicEff` macro, we could never begin to consider all possible candidate choice sets. Furthermore, with the linear arrangement approach, we could not create a design with six choice sets since the minimum size is $2 \times 15 + 1 = 31$. Now consider the alternative-swapping algorithm. It uses at most a candidate set with only 244 observations ($3^5 + 1$). From it, every possible choice set can potentially be constructed, although the macro only considers a tiny fraction of the possibilities. Hence, alternative swapping usually finds a better design, because the candidate set does not limit it.

Both uses of the `%ChoicEff` macro have the advantage that they are explicitly minimizing the variances of the parameter estimates given a model and a $\boldsymbol{\beta}$ vector. They can be used to produce smaller, more specialized, and better designs. However, if the $\boldsymbol{\beta}$ vector or model is badly misspecified, the designs could be horrible. How badly do things have to be misspecified before you will have problems? Who knows. More research is needed. In contrast, the linear model `%MktEx` approach is very conservative and safe in that it should let you specify a very general model and still produce estimable parameters. The cost is you might be using many more choice sets than you need, particularly for nonbranded generic attributes. If you really have some information about your parameters, you should use them to produce a smaller and better design. However, if you have little or no information about parameters and if you anticipate specifying very general models like mother logit, then you probably want to use the linear arrangement approach.

Initial Designs

This section illustrates some design strategies that involve improving on or augmenting initial designs. We will not actually use any designs from this section.

Improving an Existing Design

Sometimes, it is useful to try to improve an existing design. In this example, we use the `%MktEx` macro to create a design in 80 runs for 25 four-level factors. In the next step, we specify `init=`, and the macro goes straight into the design refinement history seeking to refine the input design. You might want to do this, for example, whenever you have a good, but not 100% *D*-efficient design, and you are willing to wait a few minutes to see if the macro can make it any better. The following steps generate the design:

```
title 'Try to Improve an Existing Design';

%mktex(4 ** 25, n=80, seed=368)

%mktex(4 ** 25,                  /* 25 four-level factors             */
       n=80,                     /* number of runs                    */
       init=design,              /* initial design                    */
       seed=306,                 /* random number seed                */
       maxtime=20)               /* maximum time in minutes to work   */
```

The *D*-efficiency of the final design from the first step is as follows:

```
                          Try to Improve an Existing Design

                                The OPTEX Procedure

                                                                Average
                                                               Prediction
      Design                                                    Standard
      Number     D-Efficiency     A-Efficiency     G-Efficiency       Error
      ------------------------------------------------------------------------
           1          91.4106          83.9583          97.6073          0.9747
```

This is a large problem—one in which the `maxtime=` option might cause the macro to stop before it reaches the maximum number of iterations. Running a second refinement step might help improve the design by adding a few more iterations.

The results from the second step are as follows:

```
                   Design Refinement History

                                 Current           Best
Design    Row,Col  D-Efficiency  D-Efficiency  Notes
----------------------------------------------------------
     0    Initial       91.4106       91.4106  Ini

     1      Start       90.0771                Pre,Mut,Ann
     1        End       91.3476

     2      Start       88.8927                Pre,Mut,Ann
     2     36  12       91.4181       91.4181
     2     56   6       91.4285       91.4285
     2      7  17       91.4372       91.4372
     2     13  10       91.4373       91.4373
     2     23  18       91.4404       91.4404
     2     17  16       91.4445       91.4445
     2     34   6       91.4572       91.4572
     2     56  19       91.4673       91.4673
     2     56  21       91.4768       91.4768
     2     77   1       91.4821       91.4821
     2     23  18       91.4827       91.4827
     2     48   3       91.4848       91.4848
     2     48   9       91.4863       91.4863
     2     40  18       91.4863       91.4863
     2        End       91.4863
     .
     .
     .

     6      Start       90.2194                Pre,Mut,Ann
     6     63  19       91.5811       91.5811
     6     68  18       91.5835       91.5835
     6        End       91.5751
```

```
            7      Start       89.4607                    Pre,Mut,Ann
            7     25   4       91.5851       91.5851
            7     34   7       91.5902       91.5902
            7     47   2       91.5913       91.5913
            7     48  14       91.5930       91.5930
            7     56   4       91.5955       91.5955
            7     56  15       91.5999       91.5999
            7     60   6       91.6142       91.6142
            7     68   7       91.6172       91.6172
            7     78   5       91.6172       91.6172
            7     13  21       91.6249       91.6249
            7     18  19       91.6249       91.6249
            7     43  10       91.6249       91.6249
            7     48  14       91.6282       91.6282
            7     50  22       91.6408       91.6408
            7     61   4       91.6417       91.6417
            7     80  15       91.6430       91.6430
            7     46  12       91.6430       91.6430
            7     48   6       91.6430       91.6430
            7        End       91.6430
            .
            .
            .
           10      Start       89.8707                    Pre,Mut,Ann
           10        End       91.6629
            .
            .
            .
                   Try to Improve an Existing Design

                          The OPTEX Procedure

                                                                 Average
                                                               Prediction
      Design                                                    Standard
      Number     D-Efficiency     A-Efficiency     G-Efficiency       Error
      ------------------------------------------------------------------------
           1        91.7082          83.9853          97.5951         0.9747
```

The macro skips the normal first steps, algorithm search and design search, and goes straight into the design refinement search. In this example a small improvement is found, although often, no improvement is found.

When Some Choice Sets are Fixed in Advance

Sometimes certain runs or choice sets are fixed in advance and must be included in the design. The `%MktEx` macro can be used to efficiently augment a starting design with other choice sets. Suppose that you can make a choice design from the L_{36} $(2^{11}3^{12})$. In addition, you want to optimally add four more choice sets to use as holdouts. First we look at how to do this using the `fixed=` option. This option can be used for fairly general design augmentation and refinement problems. On page 587, we see an easier way to handle this particular problem using the `holdouts=` option.

You can create the design in 36 runs as before. Next, a DATA step is used to add a flag variable `f` that has values of 1 for the original 36 runs. In addition, four more runs are added (just copies of the last run) but with a flag value of missing. When this variable is specified in the `fixed=f` option, it indicates that the first 36 runs of the `init=init` design are *fixed*—they must not change. The remaining 4 runs are to be randomly initialized and optimally refined to maximize the *D*-efficiency of the overall 40-run design. We specify `options=nosort` so that the additional runs stay at the end of the design. The following steps generate and display the design:

```
title 'Augment a Design';

%mktex(n=36, seed=292)

data init;
   set randomized end = eof;
   f = 1;
   output;
   if eof then do;
      f = .;
      do i = 1 to 4; output; end;
      drop i;
      end;
   run;

proc print; run;


%mktex(2 ** 11 3 ** 12,           /* 11 2-level and 12 3-level factors    */
       n=40,                      /* 40 runs in final design              */
       init=init,                 /* initial design                       */
       fixed=f,                   /* f var flags runs that cannot change  */
       seed=513,                  /* random number seed                   */
       options=nosort)            /* do not sort output design            */

proc print; run;
```

The initial design is as follows:

```
                         Augment a Design

 O                             x  x  x  x  x  x  x  x  x  x  x  x  x  x
 b  x  x  x  x  x  x  x  x  x  1  1  1  1  1  1  1  1  1  1  2  2  2  2
 s  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5  6  7  8  9  0  1  2  3  f

 1  2  1  2  2  2  2  1  1  1  2  1  2  3  1  1  1  2  3  2  3  1  3  2  1
 2  2  1  1  2  2  1  2  1  2  1  2  3  1  2  1  3  2  3  1  2  2  3  1  1
 3  2  2  1  1  1  1  1  1  1  2  2  1  2  3  1  1  2  2  3  2  3  3  3  1
 4  1  1  1  1  2  1  1  2  2  2  1  1  2  3  2  3  2  3  2  1  2  1  2  1
 5  1  2  1  2  2  2  1  2  1  1  2  1  1  2  2  2  1  3  3  3  3  3  2  1
 6  2  1  1  2  2  1  2  1  2  1  2  1  2  1  3  2  1  2  2  3  1  1  3  1
 7  1  2  2  1  2  1  2  1  1  1  1  3  3  2  2  1  1  2  2  1  2  3  3  1
 8  2  2  1  1  1  1  1  1  1  2  2  2  3  2  3  3  1  1  1  3  2  1  2  1
 9  2  2  2  2  1  1  1  2  2  1  1  2  2  2  3  3  3  3  3  1  1  3  3  1
10  2  2  2  2  1  1  1  2  2  1  1  1  1  3  1  1  1  1  2  3  2  2  1  1
11  1  1  2  1  1  2  1  1  2  1  2  3  2  2  2  1  2  1  3  3  1  1  1  1
12  2  2  2  1  2  2  2  2  2  2  2  1  3  3  2  3  3  2  1  3  1  3  1  1
13  1  2  1  2  2  2  1  2  1  1  2  3  3  3  3  3  2  1  2  2  1  2  3  1
14  1  1  2  1  1  2  1  1  2  1  2  1  3  1  1  3  1  3  1  1  3  2  3  1
15  2  1  1  1  1  2  2  2  1  1  1  1  3  2  3  1  3  3  2  2  3  1  1  1
16  1  2  2  1  2  1  2  1  1  1  1  2  2  3  3  2  2  3  1  3  3  2  1  1
17  2  1  1  1  1  2  2  2  1  1  1  3  2  3  1  2  1  1  1  1  1  3  2  1
18  1  1  1  1  2  1  1  2  2  2  1  3  1  1  3  1  3  1  1  3  3  3  3  1
19  2  1  2  2  2  2  1  1  1  2  1  1  2  2  2  2  3  1  1  2  2  2  3  1
20  2  1  1  1  1  2  2  2  1  1  1  2  1  1  2  3  2  2  3  3  2  2  3  1
21  1  2  1  2  1  2  2  1  2  2  1  1  3  1  3  2  2  1  3  1  2  3  1  1
22  1  1  2  1  1  2  1  1  2  1  2  2  1  3  3  2  3  2  2  2  2  3  2  1
23  2  2  2  1  2  2  2  2  2  2  2  3  2  1  3  1  1  3  3  2  2  2  2  1
24  1  2  2  1  2  1  2  1  1  1  1  1  1  1  1  3  3  1  3  2  1  1  2  1
25  1  1  2  2  1  1  2  2  1  2  2  1  1  2  3  1  2  2  1  1  1  2  2  1
26  1  1  2  2  1  1  2  2  1  2  2  2  2  1  2  3  1  1  2  2  3  3  1  1
27  1  1  2  2  1  1  2  2  1  2  2  3  3  3  1  2  3  3  3  3  2  1  3  1
28  1  2  1  2  2  2  1  2  1  1  2  2  2  1  1  1  3  2  1  1  2  1  1  1
29  2  1  2  2  2  2  1  1  1  2  1  3  1  3  3  3  1  2  3  1  3  1  1  1
30  2  2  1  1  1  1  1  1  1  2  2  3  1  1  2  2  3  3  2  1  1  2  1  1
31  1  2  1  2  1  2  2  1  2  2  1  2  1  3  2  1  1  3  1  2  1  1  3  1
32  2  1  1  2  2  1  2  1  2  1  2  2  3  3  2  1  3  1  3  1  3  2  2  1
33  2  2  2  2  1  1  1  2  2  1  1  3  3  1  2  2  2  2  1  2  3  1  2  1
34  1  2  1  2  1  2  2  1  2  2  1  3  2  2  1  3  3  2  2  3  3  2  2  1
35  2  2  2  1  2  2  2  2  2  2  2  2  1  2  1  2  2  1  2  1  3  1  3  1
36  1  1  1  1  2  1  1  2  2  2  1  2  3  2  1  2  1  2  3  2  1  2  1  1
37  1  1  1  1  2  1  1  2  2  2  1  2  3  2  1  2  1  2  3  2  1  2  1  .
38  1  1  1  1  2  1  1  2  2  2  1  2  3  2  1  2  1  2  3  2  1  2  1  .
39  1  1  1  1  2  1  1  2  2  2  1  2  3  2  1  2  1  2  3  2  1  2  1  .
40  1  1  1  1  2  1  1  2  2  2  1  2  3  2  1  2  1  2  3  2  1  2  1  .
```

The iteration history for the augmentation is as follows:

```
                                  Augment a Design

                              Design Refinement History

                             Current          Best
Design    Row,Col  D-Efficiency  D-Efficiency  Notes
----------------------------------------------------------
     0    Initial       97.0559       97.0559  Ini

     1      Start       97.0788       97.0788  Pre,Mut,Ann
     1     37   5       97.0805       97.0805
     1     37   8       97.0816       97.0816
     1     37   9       97.1049       97.1049
     1     37  11       97.1133       97.1133
     1     37  13       97.1177       97.1177
     1     38   1       97.1410       97.1410
     1     38   2       97.1605       97.1605
     1     38   3       97.1729       97.1729
     1     38   4       97.1822       97.1822
     1     38   6       97.1905       97.1905
     1     38   8       97.1944       97.1944
     1     39   2       97.1991       97.1991
     1     39  13       97.2007       97.2007
     1     39  19       97.2007       97.2007
     1     40   9       97.2007       97.2007
     1     40  10       97.2007       97.2007
     1     37  18       97.2023       97.2023
     1     37   3       97.2028       97.2028
     1     37   4       97.2028       97.2028
     1     37  23       97.2043       97.2043
     1        End       97.2043

     2      Start       97.2043       97.2043  Pre,Mut,Ann
     2     40  21       97.2043       97.2043
     2     38   2       97.2043       97.2043
     2     40   9       97.2043       97.2043
     2     40  21       97.2043       97.2043
     2        End       97.2043

     3      Start       97.2043       97.2043  Pre,Mut,Ann
     3        End       97.2043

     4      Start       97.2002                Pre,Mut,Ann
     4     39  12       97.2043       97.2043
     4     39  23       97.2043       97.2043
     4     39  16       97.2043       97.2043
     4        End       97.2043
```

```
                  5      Start       97.2043       97.2043  Pre,Mut,Ann
                  5     37   3       97.2043       97.2043
                  5     37  15       97.2043       97.2043
                  5        End       97.2043

                  6      Start       97.2043       97.2043  Pre,Mut,Ann
                  6     40   1       97.2043       97.2043
                  6     39  16       97.2043       97.2043
                  6     38  16       97.2043       97.2043
                  6        End       97.2043
NOTE: Stopping since it appears that no improvement is possible.
```

Notice that the macro goes straight into the design refinement stage. Also notice that in the iteration history, only rows 37 through 40 are changed. The design is as follows:

```
                                 Augment a Design

     O                                        x  x  x  x  x  x  x  x  x  x  x  x  x  x
     b  x  x  x  x  x  x  x  x  x  1  1  1  1  1  1  1  1  1  1  2  2  2  2
     s  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5  6  7  8  9  0  1  2  3  f

     1  2  1  2  2  2  2  1  1  1  2  1  2  3  1  1  1  2  3  2  3  1  3  2  1
     2  2  1  1  2  2  1  2  1  2  1  2  3  1  2  1  3  2  3  1  2  2  3  1  1
     3  2  2  1  1  1  1  1  1  1  2  2  1  2  3  1  1  2  2  3  2  3  3  3  1
     4  1  1  1  1  2  1  1  2  2  2  1  1  2  3  2  3  2  3  2  1  2  1  2  1
     5  1  2  1  2  2  2  1  2  1  1  2  1  1  2  2  2  1  3  3  3  3  3  2  1
     6  2  1  1  2  2  1  2  1  2  1  2  1  2  1  3  2  1  2  2  3  1  1  3  1
     7  1  2  2  1  2  1  2  1  1  1  1  3  3  2  2  1  1  2  2  1  2  3  3  1
     8  2  2  1  1  1  1  1  1  1  2  2  2  3  2  3  3  1  1  1  3  2  1  2  1
     9  2  2  2  2  1  1  1  2  2  1  1  2  2  2  3  3  3  3  3  1  1  3  3  1
    10  2  2  2  2  1  1  1  2  2  1  1  1  1  3  1  1  1  1  2  3  2  2  1  1
    11  1  1  2  1  1  2  1  1  2  1  2  3  2  2  2  1  2  1  3  3  1  1  1  1
    12  2  2  2  1  2  2  2  2  2  2  2  1  3  3  2  3  3  2  1  3  1  3  1  1
    13  1  2  1  2  2  2  1  2  1  1  2  3  3  3  3  3  2  1  2  2  1  2  3  1
    14  1  1  2  1  1  2  1  1  2  1  2  1  3  1  1  3  1  3  1  1  3  2  3  1
    15  2  1  1  1  1  2  2  2  1  1  1  1  3  2  3  1  3  3  2  2  3  1  1  1
    16  1  2  2  1  2  1  2  1  1  1  1  2  2  3  3  2  2  3  1  3  3  2  1  1
    17  2  1  1  1  1  2  2  2  1  1  1  3  2  3  1  2  1  1  1  1  1  3  2  1
    18  1  1  1  1  2  1  1  2  2  2  1  3  1  1  3  1  3  1  1  3  3  3  3  1
    19  2  1  2  2  2  2  1  1  1  2  1  1  2  2  2  2  3  1  1  2  2  2  3  1
    20  2  1  1  1  1  2  2  2  1  1  1  2  1  1  2  3  2  2  3  3  2  2  3  1
```

```
   21  1  2  1  2  1  2  2  1  2  2  1  1  3  1  3  2  2  1  3  1  2  3  1  1
   22  1  1  2  1  1  2  1  1  2  1  2  2  1  3  3  2  3  2  2  2  2  3  2  1
   23  2  2  2  1  2  2  2  2  2  2  2  3  2  1  3  1  1  3  3  2  2  2  2  1
   24  1  2  2  1  2  1  2  1  1  1  1  1  1  1  1  3  3  1  3  2  1  1  2  1
   25  1  1  2  2  1  1  2  2  1  2  2  1  1  2  3  1  2  2  1  1  1  2  2  1
   26  1  1  2  2  1  1  2  2  1  2  2  2  2  1  2  3  1  1  2  2  3  3  1  1
   27  1  1  2  2  1  1  2  2  1  2  2  3  3  3  1  2  3  3  3  3  2  1  3  1
   28  1  2  1  2  2  2  1  2  1  1  2  2  2  1  1  1  3  2  1  1  2  1  1  1
   29  2  1  2  2  2  2  1  1  1  2  1  3  1  3  3  3  1  2  3  1  3  1  1  1
   30  2  2  1  1  1  1  1  1  1  2  2  3  1  1  2  2  3  3  2  1  1  2  1  1
   31  1  2  1  2  1  2  2  1  2  2  1  2  1  3  2  1  1  3  1  2  1  1  3  1
   32  2  1  1  2  2  1  2  1  2  1  2  2  3  3  2  1  3  1  3  1  3  2  2  1
   33  2  2  2  2  1  1  1  2  2  1  1  3  3  1  2  2  2  2  1  2  3  1  2  1
   34  1  2  1  2  1  2  2  1  2  2  1  3  2  2  1  3  3  2  2  3  3  2  2  1
   35  2  2  2  1  2  2  2  2  2  2  2  2  1  2  1  2  2  1  2  1  3  1  3  1
   36  1  1  1  1  2  1  1  2  2  2  1  2  3  2  1  2  1  2  3  2  1  2  1  1
   37  1  2  2  2  2  1  1  2  1  1  1  3  2  3  3  2  1  2  1  3  1  3  1  .
   38  2  2  2  1  2  1  2  2  1  1  1  2  2  1  2  3  3  3  1  1  3  2  3  .
   39  1  2  2  2  2  2  2  1  2  2  2  3  3  3  1  2  1  3  2  1  3  2  3  .
   40  2  1  1  1  1  1  1  2  2  2  1  2  1  2  3  2  1  3  2  1  1  2  1  .
```

The last four rows are the holdouts.

The following steps do the same thing only using the `holdouts=4` option instead:

```
   title 'Augment a Design';

   %mktex(n=36, seed=292)

   %mktex(2 ** 11 3 ** 12,            /* 11 2-level and 12 3-level factors    */
          n=40,                       /* 40 runs in final design              */
          init=randomized,            /* initial design with 36 runs          */
          holdouts=4,                 /* add four holdouts                    */
          seed=513,                   /* random number seed                   */
          options=nosort)             /* do not sort output design            */

   proc print data=design(firstobs=37); run;
```

The holdout observations, which are the same as we saw previously, are as follows:

```
                                  Augment a Design

     O                                       x  x  x  x  x  x  x  x  x  x  x  x  x  x
     b  x  x  x  x  x  x  x  x  x  1  1  1  1  1  1  1  1  1  1  2  2  2  2
     s  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5  6  7  8  9  0  1  2  3  w

    37  1  2  2  2  2  1  1  2  1  1  1  3  2  3  3  2  1  2  1  3  1  3  1  .
    38  2  2  2  1  2  1  2  2  1  1  1  2  2  1  2  3  3  3  1  1  3  2  3  .
    39  1  2  2  2  2  2  2  1  2  2  2  3  3  3  1  2  1  3  2  1  3  2  3  .
    40  2  1  1  1  1  1  1  2  2  2  1  2  1  2  3  2  1  3  2  1  1  2  1  .
```

The %MktEx macro provides another way to use initial designs. The initial design can indicate that part of the design is fixed and cannot change and a different part should be randomly initialized and can change. The initial design can have three types of values:

- positive integers are fixed and constant and do not change throughout the course of the iterations.
- zero and missing values are replaced by random values at the start of each new design search and can change throughout the course of the iterations.
- negative values are replaced by their absolute value at the start of each new design attempt and can change throughout the course of the iterations.

Returning to the example of making the design 4^{25} in 80 runs, we could do it in two steps. The maximum number of four-level factors in 80 runs is 11. If it is important that some factors be orthogonal, we could first make an orthogonal array with 11 four-level factors and then append 14 more nonorthogonal-factors. The following steps create the design:

```
title 'Differential Design Initialization';

%mktex(4 ** 11, n=80)

data init;
   set design;
   retain x12-x25 .;
   run;

%mktex(4 ** 25,                    /* 25 four-level factors             */
       n=80,                       /* 80 runs                           */
       init=init,                  /* initial design                    */
       seed=472)                   /* random number seed                */

%mkteval;
```

The initial design consists of the orthogonal array with 11 columns followed by 14 more columns that are all missing. When %MktEx sees missing values in the initial design, it holds all the nonmissing values fixed. Then it randomly replaces the missing values and uses the coordinate-exchange algorithm to refine the last columns. The final design is slightly less D-efficient than we saw previously, but the first 11 columns are orthogonal. The remaining columns are all slightly correlated with themselves and with the first columns. The final efficiency table is as follows:

```
                    Differential Design Initialization

                                                                   Average
                                                                  Prediction
       Design                                                      Standard
       Number     D-Efficiency     A-Efficiency     G-Efficiency     Error
       ------------------------------------------------------------------------
            1        89.4216          78.9807          97.7767          0.9747
```

The canonical correlations are as follows:

```
                    Differential Design Initialization
                 Canonical Correlations Between the Factors
           There are 0 Canonical Correlations Greater Than 0.316

          x1   x2   x3   x4   x5   x6   x7   x8   x9   x10  x11  x12  x13

     x1   1    0    0    0    0    0    0    0    0    0    0    0.15 0.17
     x2   0    1    0    0    0    0    0    0    0    0    0    0.17 0.11
     x3   0    0    1    0    0    0    0    0    0    0    0    0.20 0.13
     x4   0    0    0    1    0    0    0    0    0    0    0    0.16 0.12
     x5   0    0    0    0    1    0    0    0    0    0    0    0.16 0.18
     x6   0    0    0    0    0    1    0    0    0    0    0    0.12 0.12
     x7   0    0    0    0    0    0    1    0    0    0    0    0.18 0.18
     x8   0    0    0    0    0    0    0    1    0    0    0    0.16 0.12
     x9   0    0    0    0    0    0    0    0    1    0    0    0.15 0.14
     x10  0    0    0    0    0    0    0    0    0    1    0    0.12 0.18
     x11  0    0    0    0    0    0    0    0    0    0    1    0.16 0.12
     x12  0.15 0.17 0.20 0.16 0.16 0.12 0.18 0.16 0.15 0.12 0.16 1    0.10
     x13  0.17 0.11 0.13 0.12 0.18 0.12 0.18 0.12 0.14 0.18 0.12 0.10 1
     x14  0.18 0.14 0.10 0.14 0.19 0.20 0.18 0.19 0.14 0.15 0.11 0.16 0.15
     x15  0.16 0.25 0.15 0.18 0.17 0.17 0.15 0.12 0.24 0.13 0.16 0.14 0.13
     x16  0.10 0.23 0.14 0.15 0.17 0.16 0.15 0.13 0.10 0.16 0.21 0.12 0.11
     x17  0.20 0.20 0.15 0.07 0.12 0.17 0.16 0.12 0.13 0.20 0.14 0.12 0.11
     x18  0.09 0.18 0.10 0.16 0.15 0.13 0.13 0.17 0.10 0.15 0.14 0.15 0.09
     x19  0.17 0.18 0.17 0.16 0.13 0.17 0.19 0.21 0.15 0.15 0.16 0.14 0.15
     x20  0.23 0.09 0.11 0.14 0.17 0.21 0.25 0.27 0.06 0.17 0.15 0.14 0.11
     x21  0.12 0.09 0.15 0.17 0.17 0.15 0.20 0.14 0.20 0.17 0.16 0.18 0.12
     x22  0.17 0.14 0.07 0.18 0.15 0.12 0.11 0.16 0.17 0.09 0.17 0.12 0.16
     x23  0.19 0.15 0.15 0.18 0.09 0.10 0.16 0.16 0.12 0.16 0.16 0.13 0.12
     x24  0.18 0.16 0.14 0.20 0.09 0.22 0.13 0.13 0.18 0.13 0.18 0.14 0.11
     x25  0.14 0.18 0.09 0.13 0.19 0.18 0.17 0.19 0.16 0.20 0.14 0.15 0.12
     x14    x15    x16    x17    x18    x19    x20    x21    x22    x23    x24    x25

x1   0.18   0.16   0.10   0.20   0.09   0.17   0.23   0.12   0.17   0.19   0.18   0.14
x2   0.14   0.25   0.23   0.20   0.18   0.18   0.09   0.09   0.14   0.15   0.16   0.18
x3   0.10   0.15   0.14   0.15   0.10   0.17   0.11   0.15   0.07   0.15   0.14   0.09
x4   0.14   0.18   0.15   0.07   0.16   0.16   0.14   0.17   0.18   0.18   0.20   0.13
x5   0.19   0.17   0.17   0.12   0.15   0.13   0.17   0.17   0.15   0.09   0.09   0.19
x6   0.20   0.17   0.16   0.17   0.13   0.17   0.21   0.15   0.12   0.10   0.22   0.18
x7   0.18   0.15   0.15   0.16   0.13   0.19   0.25   0.20   0.11   0.16   0.13   0.17
x8   0.19   0.12   0.13   0.12   0.17   0.21   0.27   0.14   0.16   0.16   0.13   0.19
x9   0.14   0.24   0.10   0.13   0.10   0.15   0.06   0.20   0.17   0.12   0.18   0.16
x10  0.15   0.13   0.16   0.20   0.15   0.15   0.17   0.17   0.09   0.16   0.13   0.20
x11  0.11   0.16   0.21   0.14   0.14   0.16   0.15   0.16   0.17   0.16   0.18   0.14
x12  0.16   0.14   0.12   0.12   0.15   0.14   0.14   0.18   0.12   0.13   0.14   0.15
x13  0.15   0.13   0.11   0.11   0.09   0.15   0.11   0.12   0.16   0.12   0.11   0.12
x14  1      0.22   0.14   0.12   0.12   0.21   0.13   0.13   0.13   0.20   0.18   0.12
x15  0.22   1      0.12   0.09   0.19   0.09   0.09   0.13   0.10   0.15   0.16   0.13
x16  0.14   0.12   1      0.15   0.14   0.14   0.18   0.13   0.12   0.10   0.17   0.18
x17  0.12   0.09   0.15   1      0.09   0.16   0.18   0.09   0.15   0.12   0.18   0.12
x18  0.12   0.19   0.14   0.09   1      0.14   0.13   0.20   0.15   0.13   0.14   0.17
x19  0.21   0.09   0.14   0.16   0.14   1      0.13   0.11   0.10   0.11   0.16   0.12
x20  0.13   0.09   0.18   0.18   0.13   0.13   1      0.10   0.11   0.11   0.18   0.11
x21  0.13   0.13   0.13   0.09   0.20   0.11   0.10   1      0.21   0.15   0.23   0.09
x22  0.13   0.10   0.12   0.15   0.15   0.10   0.11   0.21   1      0.19   0.18   0.22
x23  0.20   0.15   0.10   0.12   0.13   0.11   0.11   0.15   0.19   1      0.13   0.13
x24  0.18   0.16   0.17   0.18   0.14   0.16   0.18   0.23   0.18   0.13   1      0.12
x25  0.12   0.13   0.18   0.12   0.17   0.12   0.11   0.09   0.22   0.13   0.12   1
```

The one-way frequencies are as follows:

```
                    Differential Design Initialization
                          Summary of Frequencies
          There are 0 Canonical Correlations Greater Than 0.316
                    * - Indicates Unequal Frequencies

                             Frequencies

                x1          20 20 20 20
                x2          20 20 20 20
                x3          20 20 20 20
                x4          20 20 20 20
                x5          20 20 20 20
                x6          20 20 20 20
                x7          20 20 20 20
                x8          20 20 20 20
                x9          20 20 20 20
                x10         20 20 20 20
                x11         20 20 20 20
```

```
             *    x12        19 22 19 20
             *    x13        21 19 20 20
             *    x14        18 20 21 21
             *    x15        18 21 22 19
             *    x16        19 18 22 21
             *    x17        21 19 18 22
             *    x18        21 20 20 19
             *    x19        21 18 18 23
             *    x20        20 20 21 19
             *    x21        18 20 23 19
             *    x22        20 18 22 20
             *    x23        19 20 21 20
             *    x24        23 17 20 20
             *    x25        21 19 19 21
```

We could run one more refinement on this design and force `x12-x25` to be balanced. We need to make a new initial design using our current design. This time, we make `x12-x25` negative. Then `x1-x11` do not change, the absolute values of `x12-x25` are used as initial values, but `x12-x25` are still be allowed to change. Then we can use the `balance=1` option with `%MktEx` to make a design that is better balanced. Usually, when you are creating a design, you should specify `mintry=` with `balance=`, however, since we are refining not creating a design it is not necessary. The following steps create the design:

```
data init(drop=j);
   set design;
   array x[25];
   do j = 12 to 25; x[j] = -x[j]; end;
   run;

%mktex(4 ** 25,                   /* 25 four-level factors            */
       n=80,                      /* 80 runs                          */
       init=init,                 /* initial design                   */
       seed=472,                  /* random number seed               */
       balance=1)                 /* require near perfect balance     */

%mkteval;
```

The final *D*-efficiency, which again, is a bit lower than we saw previously, is as follows:

```
                        Differential Design Initialization

                                                                Average
                                                               Prediction
    Design                                                      Standard
    Number     D-Efficiency     A-Efficiency     G-Efficiency     Error
    ------------------------------------------------------------------------
         1          86.1917          71.1301          97.9379       0.9747
```

The canonical correlations are as follows:

```
                  Differential Design Initialization
                Canonical Correlations Between the Factors
            There are 0 Canonical Correlations Greater Than 0.316

       x1   x2   x3   x4   x5   x6   x7   x8   x9   x10  x11  x12  x13

   x1  1    0    0    0    0    0    0    0    0    0    0    0.20 0.15
   x2  0    1    0    0    0    0    0    0    0    0    0    0.15 0.13
   x3  0    0    1    0    0    0    0    0    0    0    0    0.15 0.12
   x4  0    0    0    1    0    0    0    0    0    0    0    0.20 0.15
   x5  0    0    0    0    1    0    0    0    0    0    0    0.20 0.24
   x6  0    0    0    0    0    1    0    0    0    0    0    0.13 0.15
   x7  0    0    0    0    0    0    1    0    0    0    0    0.17 0.19
   x8  0    0    0    0    0    0    0    1    0    0    0    0.13 0.15
   x9  0    0    0    0    0    0    0    0    1    0    0    0.12 0.16
   x10 0    0    0    0    0    0    0    0    0    1    0    0.13 0.17
   x11 0    0    0    0    0    0    0    0    0    0    1    0.20 0.13
   x12 0.20 0.15 0.15 0.20 0.20 0.13 0.17 0.13 0.12 0.13 0.20 1    0.14
   x13 0.15 0.13 0.12 0.15 0.24 0.15 0.19 0.15 0.16 0.17 0.13 0.14 1
   x14 0.20 0.10 0.09 0.13 0.18 0.22 0.17 0.16 0.13 0.16 0.09 0.20 0.15
   x15 0.21 0.23 0.22 0.25 0.20 0.24 0.17 0.15 0.22 0.13 0.22 0.18 0.16
   x16 0.09 0.20 0.17 0.18 0.12 0.18 0.17 0.17 0.10 0.16 0.20 0.09 0.14
   x17 0.20 0.23 0.15 0.15 0.09 0.22 0.20 0.10 0.15 0.17 0.15 0.15 0.15
   x18 0.10 0.23 0.10 0.16 0.15 0.15 0.18 0.17 0.10 0.15 0.15 0.15 0.12
   x19 0.17 0.20 0.18 0.14 0.17 0.15 0.23 0.16 0.12 0.19 0.17 0.16 0.25
   x20 0.26 0.10 0.10 0.17 0.17 0.17 0.24 0.23 0.09 0.17 0.19 0.12 0.10
   x21 0.20 0.13 0.20 0.19 0.19 0.19 0.20 0.20 0.15 0.14 0.17 0.14 0.14
   x22 0.17 0.16 0.10 0.23 0.14 0.16 0.10 0.16 0.15 0.10 0.14 0.15 0.15
   x23 0.24 0.18 0.14 0.18 0.09 0.10 0.15 0.20 0.13 0.22 0.19 0.19 0.12
   x24 0.24 0.20 0.27 0.18 0.15 0.29 0.12 0.12 0.18 0.20 0.17 0.20 0.14
   x25 0.20 0.16 0.09 0.15 0.23 0.19 0.17 0.18 0.17 0.22 0.15 0.16 0.12
```

```
      x14   x15   x16   x17   x18   x19   x20   x21   x22   x23   x24   x25

x1    0.20  0.21  0.09  0.20  0.10  0.17  0.26  0.20  0.17  0.24  0.24  0.20
x2    0.10  0.23  0.20  0.23  0.23  0.20  0.10  0.13  0.16  0.18  0.20  0.16
x3    0.09  0.22  0.17  0.15  0.10  0.18  0.10  0.20  0.10  0.14  0.27  0.09
x4    0.13  0.25  0.18  0.15  0.16  0.14  0.17  0.19  0.23  0.18  0.18  0.15
x5    0.18  0.20  0.12  0.09  0.15  0.17  0.17  0.19  0.14  0.09  0.15  0.23
x6    0.22  0.24  0.18  0.22  0.15  0.15  0.17  0.19  0.16  0.10  0.29  0.19
x7    0.17  0.17  0.17  0.20  0.18  0.23  0.24  0.20  0.10  0.15  0.12  0.17
x8    0.16  0.15  0.17  0.10  0.17  0.16  0.23  0.20  0.16  0.20  0.12  0.18
x9    0.13  0.22  0.10  0.15  0.10  0.12  0.09  0.15  0.15  0.13  0.18  0.17
x10   0.16  0.13  0.16  0.17  0.15  0.19  0.17  0.14  0.10  0.22  0.20  0.22
x11   0.09  0.22  0.20  0.15  0.15  0.17  0.19  0.17  0.14  0.19  0.17  0.15
x12   0.20  0.18  0.09  0.15  0.15  0.16  0.12  0.14  0.15  0.19  0.20  0.16
x13   0.15  0.16  0.14  0.15  0.12  0.25  0.10  0.14  0.15  0.12  0.14  0.12
x14   1     0.20  0.20  0.12  0.18  0.24  0.10  0.25  0.14  0.18  0.17  0.13
x15   0.20  1     0.15  0.15  0.15  0.12  0.09  0.15  0.14  0.12  0.21  0.18
x16   0.20  0.15  1     0.15  0.15  0.12  0.14  0.13  0.09  0.15  0.23  0.17
x17   0.12  0.15  0.15  1     0.20  0.15  0.19  0.24  0.17  0.14  0.26  0.19
x18   0.18  0.15  0.15  0.20  1     0.17  0.17  0.22  0.15  0.09  0.17  0.22
x19   0.24  0.12  0.12  0.15  0.17  1     0.09  0.15  0.17  0.14  0.15  0.13
x20   0.10  0.09  0.14  0.19  0.17  0.09  1     0.12  0.20  0.18  0.10  0.09
x21   0.25  0.15  0.13  0.24  0.22  0.15  0.12  1     0.15  0.12  0.28  0.12
x22   0.14  0.14  0.09  0.17  0.15  0.17  0.20  0.15  1     0.22  0.17  0.27
x23   0.18  0.12  0.15  0.14  0.09  0.14  0.18  0.12  0.22  1     0.21  0.18
x24   0.17  0.21  0.23  0.26  0.17  0.15  0.10  0.28  0.17  0.21  1     0.14
x25   0.13  0.18  0.17  0.19  0.22  0.13  0.09  0.12  0.27  0.18  0.14  1
```

The one-way frequencies, which are now all perfect, are as follows:

```
                        Differential Design Initialization

                              Summary of Frequencies
                There are 0 Canonical Correlations Greater Than 0.316
                        * - Indicates Unequal Frequencies

                              Frequencies

                     x1         20 20 20 20
                     x2         20 20 20 20
                     x3         20 20 20 20
                     x4         20 20 20 20
                     x5         20 20 20 20
                     x6         20 20 20 20
                     x7         20 20 20 20
                     x8         20 20 20 20
                     x9         20 20 20 20
                     x10        20 20 20 20
                     x11        20 20 20 20
                     x12        20 20 20 20
                     x13        20 20 20 20
                     x14        20 20 20 20
                     x15        20 20 20 20
                     x16        20 20 20 20
                     x17        20 20 20 20
                     x18        20 20 20 20
                     x19        20 20 20 20
                     x20        20 20 20 20
                     x21        20 20 20 20
                     x22        20 20 20 20
                     x23        20 20 20 20
                     x24        20 20 20 20
                     x25        20 20 20 20
```

You can initialize any part of the design to positive integers (fixed), any other part to zero or missing (randomly initialize and change), and any other part to negative (do not reinitialize but change is allowed). This capability gives you very flexible control over the components of your design.

Partial Profiles and Restrictions

Partial-profile designs (Chrzan and Elrod 1995) are used when there are many attributes but no more than a few of them are allowed to vary at any one time. Chrzan and Elrod show an example where respondents must choose between vacuum cleaners that vary along 20 different attributes: Brand, Price, Warranty, Horsepower, and so on. It is difficult for respondents to simultaneously evaluate that many attributes, so it is better if they are only exposed to a few at a time. Partial-profile designs have become very popular among some researchers.

This section serves two purposes. It shows ways in which partial-profile designs can be made. It also illustrates some of the important techniques available for restricting designs with the `%MktEx` macro. Partial-profile designs provide a nice context for illustrating the creation of highly-restricted designs.

Pairwise Partial-Profile Choice Design

For example, a partial-profile design for 20 two-level factors, with 5 varying at a time, with the factors that are not shown displayed with an ordinary missing value is as follows:

```
    . . . 2 . . . 1 1 1 . . . . 1 . . . . .
    2 . . . 2 . . . . . 1 . . . . . 1 . . 1
    2 . 1 . . . . 1 . . . . . 1 . . . . . 2
    . . . . . . . 2 . . . . . 2 2 . . 2 2 .
    . . . . 2 . . . . 2 . 2 . . . 2 . 2 . .
    . . . . . . . . 2 . . . . 1 . . . 2 1 1
    . . . . . . . . 1 . 2 . 2 . . . 2 . . 2
    1 . . . . 1 . . . . . . . 2 . 2 . . . 1
    . . . . . . 2 . . . . 2 1 . 2 . . . . 1
    . . 1 . 1 . . . . 2 . . . . . 1 . . . 1
    1 . 2 . . . 2 . . . . . . . . . 1 2 . .
    . 1 . . . . . . . . 2 . 1 . 1 1 . . . .
    2 . . . . . . . 2 . . 2 . . . 1 2 . . .
    . . 1 . . . . . . . . 2 2 . 1 . . . 1 .
    . . . . 2 . 2 . . . . . . 2 . 1 . . 1 .
    . 2 . 1 . . . 1 . . . . . . . . 2 . 2 .
    2 2 . . . . 1 . 1 . . 1 . . . . . . . .
    . 2 . . . . . . . 1 1 . . . . . . 1 . 2
    . 1 . 1 . 2 . 2 . . . . . . . . . 1 . .
    . . . 1 1 . . . . . 2 2 . 1 . . . . . .
    1 . . . 2 2 1 . . . . . . . 2 . . . . .
    . . . . . 1 1 2 . 1 . . . . . . 1 . . .
    . . . 2 . . . . . . . 1 2 1 . 1 . . . .
    . . . 1 . . . . 2 . . 1 . . 1 . . . . 2
    . . 2 . . 1 . . 1 . 1 . . . 2 . . . . .
    . . 2 . 1 . . 2 . . . . 1 . . 2 . . . .
    . . 2 . . . 1 1 . . 2 . . . . . . 1 . .
    . 1 . . 1 . . 1 2 . . . 2 . . . . . . .
    1 . . . . . . . 1 . . . 1 . . . . 1 2 .
    2 . . 2 1 1 . . . . . . . . . . . 2 . .
    . . . . . . . . . . . . . . 2 2 1 1 1 .
    . 1 2 2 . . . . . . . . . . . . . . 2 2
    . . 1 1 . . 1 . . . 1 . . . . 2 . . . .
    . . . . . 2 . . . 2 . . 1 2 . . 1 . . .
    . . . . . . 2 . . 2 . . . 1 1 . 2 . . .
    1 . . . . . . 2 . 2 2 . . . . . . . 1 .
    2 . 2 1 . . . . . 1 . . 2 . . . . . . .
    . 1 1 . . . . . . 1 . 1 . . . . 2 . . .
    . 2 1 2 2 . . . 2 . . . . . . . . . . .
    . . . . . 2 2 . . . 2 . . . . 2 . . 2 .
```

A design like this could be used to make a binary choice experiment. For example, the second last run has factors 2, 3, 4, 5, and 9 varying. Assume they are all yes-no factors (1 yes, 2 no). Subjects could be offered a choice between these two profiles:

```
x2 = no,   x3 = yes,  x4 = no,   x5 = no,  x9 = no
x2 = yes,  x3 = no,   x4 = yes,  x5 = yes, x9 = yes
```

The first profile came directly from the design and the second came from shifting the design: yes → no, and no → yes.

The following steps generated and displayed the partial-profile design:[*]

```
   title 'Partial Profiles';

   %mktex(3 ** 20,                       /* 20 three-level factors            */
          n=41,                          /* 41 runs                           */
          partial=5,                     /* partial profile, 5 attrs vary     */
          seed=292,                      /* random number seed                */
          maxdesigns=1)                  /* make only one design              */

   %mktlab(data=randomized, values=. 1 2, nfill=99)

   data _null_; set final(firstobs=2); put (x1-x20) (2.); run;
```

A 3^{20} design is requested in 41 runs. The three levels are yes, no, and not shown. Forty-one runs gives us 40 partial profiles and one more run with all attributes not shown (all ones in the original design before reassigning levels). When we ask for partial profiles, in this case `partial=5`, we are imposing a constraint that the number of 2's and 3's in each run equals 5 and the number of 1's equals 15. This makes the sum of the coded variables constant in each run and hence introduces a linear dependency (the sum of the coded variables is proportional to the intercept). The way we avoid having the linear dependency is by adding this additional row where all attributes are set to the not-shown level. The sum of the coded variables for this row are different than the constant sum for the other rows and hence eliminate the linear dependency we would otherwise have.

[*]Due to machine, SAS release, and macro differences, you might not get exactly the same design used in this book, but the differences should be slight.

The `%MktLab` macro reassigns the levels (1, 2, 3) to (., 1, 2) where "." means not shown. Normally, the `%MktLab` macro complains about using missing values for levels, because missing values are used in the `key=` data set as fillers when some factors have more levels than others. Encountering missing levels normally indicates an error. We can permit missing levels by specifying `nfill=99`. Then the macro considers levels of 99 to be invalid, not missing. A DATA step displays the design excluding the constant (all not shown) first row.

The next steps take this design and turn it into a partial-profile choice design. It reads each profile in the design, and outputs it. If the level is not missing, the code changes 1 to 2 and 2 to 1 and outputs the new profile. The next step uses the `%ChoicEff` macro to evaluate the design. We specify `zero=none` for now to see exactly which parameters we can estimate and which ones we cannot. This usage of the `%ChoicEff` macro is similar to what we saw in the food product example on page 509. Our choice design is specified on the `data=` option and the same data set, with just the `Set` variable kept, is specified on the `init=` option. The number of choice sets, 40 (we drop the constant choice set), number of alternatives, 2, and assumed betas, a vector of zeros, are also specified. Zero internal iterations are requested since we want a design evaluation, not an attempt to improve the design. The following steps generate and evaluate the design:

```
data des(drop=i);
   Set = _n_;
   set final(firstobs=2);
   array x[20];
   output;
   do i = 1 to 20;
      if n(x[i]) then do; if x[i] = 1 then x[i] = 2; else x[i] = 1; end;
      end;
   output;
   run;

%choiceff(data=des,                 /* candidate set of choice sets         */
          init=des(keep=set),       /* select these sets from candidates    */
          intiter=0,                /* evaluate without internal iterations */
          model=class(x1-x20 / zero=none), /* main effects, no ref levels   */
          nsets=40,                 /* number of choice sets                */
          nalts=2,                  /* number of alternatives               */
          beta=zero)                /* assumed beta vector, Ho: b=0         */

%mktdups(generic, data=best, nalts=2, factors=x1-x20)
```

The last part of the output is as follows:

```
                         Partial Profiles

           Variable                                        Standard
     n       Name        Label     Variance     DF           Error

     1       x11         x1 1       0.46962      1         0.68529
     2       x12         x1 2        .           0          .
     3       x21         x2 1       0.52778      1         0.72648
     4       x22         x2 2        .           0          .
     5       x31         x3 1       0.42989      1         0.65566
     6       x32         x3 2        .           0          .
     7       x41         x4 1       0.46230      1         0.67993
     8       x42         x4 2        .           0          .
     9       x51         x5 1       0.54615      1         0.73902
    10       x52         x5 2        .           0          .
    11       x61         x6 1       0.81069      1         0.90038
    12       x62         x6 2        .           0          .
    13       x71         x7 1       0.50135      1         0.70806
    14       x72         x7 2        .           0          .
    15       x81         x8 1       0.49753      1         0.70536
    16       x82         x8 2        .           0          .
    17       x91         x9 1       0.48632      1         0.69737
    18       x92         x9 2        .           0          .
    19       x101        x10 1      0.54529      1         0.73844
    20       x102        x10 2       .           0          .
    21       x111        x11 1      0.56975      1         0.75482
    22       x112        x11 2       .           0          .
    23       x121        x12 1      0.54158      1         0.73592
    24       x122        x12 2       .           0          .
    25       x131        x13 1      0.54817      1         0.74039
    26       x132        x13 2       .           0          .
    27       x141        x14 1      0.55059      1         0.74201
    28       x142        x14 2       .           0          .
    29       x151        x15 1      0.52638      1         0.72552
    30       x152        x15 2       .           0          .
    31       x161        x16 1      0.44403      1         0.66636
    32       x162        x16 2       .           0          .
    33       x171        x17 1      0.57751      1         0.75994
    34       x172        x17 2       .           0          .
    35       x181        x18 1      0.56915      1         0.75442
    36       x182        x18 2       .           0          .
    37       x191        x19 1      0.58340      1         0.76381
    38       x192        x19 2       .           0          .
    39       x201        x20 1      0.54343      1         0.73718
    40       x202        x20 2       .           0          .
                                                ==
                                                20
```

```
Design:          Generic
Factors:         x1-x20

x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17 x18 x19 x20
Sets w Dup Alts: 0
Duplicate Sets:  0
```

We see that one parameter is estimable for each factor and that is the parameter for the 1 or yes level. We also see that there are no duplicates. The `%ChoicEff` macro displays the following list of all redundant variables:

```
Redundant Variables:

x12 x22 x32 x42 x52 x62 x72 x82 x92 x102 x112 x122 x132 x142 x152 x162 x172 x182
x192 x202
```

We can copy and paste this list into our program and drop those terms as follows:

```
%choiceff(data=des,                 /* candidate set of choice sets         */
          init=des(keep=set),       /* select these sets from candidates    */
          intiter=0,                /* evaluate without internal iterations */
          model=class(x1-x20 / zero=none), /* main effects, no ref levels   */
          nsets=40,                 /* number of choice sets                */
          nalts=2,                  /* number of alternatives               */
                                    /* extra model terms to drop from model */
          drop=x12 x22 x32 x42 x52 x62 x72 x82 x92 x102
          x112 x122 x132 x142 x152 x162 x172 x182 x192 x202,
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

The last part of the output is as follows:

```
                              Partial Profiles

                               Final Results

                          Design                  1
                          Choice Sets            40
                          Alternatives            2
                          Parameters             20
                          Maximum Parameters     40
                          D-Efficiency       2.1648
                          D-Error            0.4619
```

```
                        Partial Profiles

          Variable                                  Standard
      n     Name      Label     Variance    DF        Error

      1     x11       x1 1       0.46962     1      0.68529
      2     x21       x2 1       0.52778     1      0.72648
      3     x31       x3 1       0.42989     1      0.65566
      4     x41       x4 1       0.46230     1      0.67993
      5     x51       x5 1       0.54615     1      0.73902
      6     x61       x6 1       0.81069     1      0.90038
      7     x71       x7 1       0.50135     1      0.70806
      8     x81       x8 1       0.49753     1      0.70536
      9     x91       x9 1       0.48632     1      0.69737
     10     x101      x10 1      0.54529     1      0.73844
     11     x111      x11 1      0.56975     1      0.75482
     12     x121      x12 1      0.54158     1      0.73592
     13     x131      x13 1      0.54817     1      0.74039
     14     x141      x14 1      0.55059     1      0.74201
     15     x151      x15 1      0.52638     1      0.72552
     16     x161      x16 1      0.44403     1      0.66636
     17     x171      x17 1      0.57751     1      0.75994
     18     x181      x18 1      0.56915     1      0.75442
     19     x191      x19 1      0.58340     1      0.76381
     20     x201      x20 1      0.54343     1      0.73718
                                            ==
                                            20
```

We could also run this using the standardized orthogonal contrast coding as follows:

```
   %choiceff(data=des,                /* candidate set of choice sets         */
             init=des(keep=set),      /* select these sets from candidates    */
             intiter=0,               /* evaluate without internal iterations */
             model=class(x1-x20 / sta),/* model with stdzd orthogonal coding   */
             nsets=40,                /* number of choice sets                */
             nalts=2,                 /* number of alternatives               */
                                      /* extra model terms to drop from model */
             options=relative,        /* display relative D-efficiency        */
             beta=zero)               /* assumed beta vector, Ho: b=0         */
```

The results are as follows:

```
                           Partial Profiles

                            Final Results

                      Design                    1
                      Choice Sets              40
                      Alternatives              2
                      Parameters               20
                      Maximum Parameters       40
                      D-Efficiency         8.6590
                      Relative D-Eff      21.6476
                      D-Error              0.1155
                      1 / Choice Sets      0.0250

                           Partial Profiles

                Variable                                     Standard
           n      Name       Label     Variance     DF         Error

           1      x11        x1 1       0.11741      1       0.34264
           2      x21        x2 1       0.13194      1       0.36324
           3      x31        x3 1       0.10747      1       0.32783
           4      x41        x4 1       0.11558      1       0.33996
           5      x51        x5 1       0.13654      1       0.36951
           6      x61        x6 1       0.20267      1       0.45019
           7      x71        x7 1       0.12534      1       0.35403
           8      x81        x8 1       0.12438      1       0.35268
           9      x91        x9 1       0.12158      1       0.34868
          10      x101       x10 1      0.13632      1       0.36922
          11      x111       x11 1      0.14244      1       0.37741
          12      x121       x12 1      0.13539      1       0.36796
          13      x131       x13 1      0.13704      1       0.37019
          14      x141       x14 1      0.13765      1       0.37101
          15      x151       x15 1      0.13159      1       0.36276
          16      x161       x16 1      0.11101      1       0.33318
          17      x171       x17 1      0.14438      1       0.37997
          18      x181       x18 1      0.14229      1       0.37721
          19      x191       x19 1      0.14585      1       0.38190
          20      x201       x20 1      0.13586      1       0.36859
                                                    ==
                                                    20
```

The output has the same model terms as previously, but the efficiencies and variances have changed. Relative *D*-efficiency is 21.6476. The largest it could be under optimal circumstances is 25 since only 5 of 20 attributes (25%) can vary.

Linear Partial-Profile Design

This section provides another example. Say that you want to make a design in 36 runs with 12 three-level factors, but you want only four factors to be considered at a time. You need to create four-level factors with one of the levels meaning not shown. You also need to ask for a design in 37 runs, because with partial profiles, one run must be all-constant. The following steps create a partial-profile design with the `%MktEx` macro using the `partial=` option:

```
title 'Partial Profiles';

%mktex(4 ** 12,                   /* 12 four-level factor                 */
       n=37,                      /* 37 runs                              */
       partial=4,                 /* partial profile, 4 attrs vary        */
       seed=462,                  /* random number seed                   */
       maxdesigns=1)              /* make only one design                 */

%mktlab(data=randomized, values=. 1 2 3, nfill=99)

proc print; run;
```

The iteration history proceeds like before, so we will not discuss it. The final *D*-efficiency is as follows:

```
                              Partial Profiles

                            The OPTEX Procedure

                                                                 Average
                                                               Prediction
    Design                                                      Standard
    Number     D-Efficiency     A-Efficiency     G-Efficiency     Error
    ------------------------------------------------------------------------
         1          49.4048          22.4346         100.0000       1.0000
```

With partial-profile designs, *D*-efficiency is typically much less than we are accustomed to seeing with other types of designs. The design is as follows:

```
                               Partial Profiles

 Obs    x1    x2    x3    x4    x5    x6    x7    x8    x9    x10    x11    x12

   1     .     .     .     .     .     .     .     .     .     .      .      .
   2     .     .     1     1     .     .     .     .     .     1      .      1
   3     .     .     1     .     .     .     2     .     1     .      2      .
   4     .     2     2     .     .     .     .     .     .     3      .      3
   5     .     .     3     3     3     .     3     .     .     .      .      .
   6     .     .     .     .     2     .     1     .     .     .      3      3
   7     1     .     .     .     2     .     .     .     .     3      1      .
   8     .     1     .     .     1     .     .     .     1     2      .      .
   9     2     .     .     .     .     .     .     3     .     .      1      3
  10     .     2     2     .     .     .     .     .     3     .      1      .
  11     2     2     .     2     .     .     1     .     .     .      .      .
  12     .     .     .     .     3     3     .     .     .     2      2      .
  13     1     .     2     .     .     2     1     .     .     .      .      .
  14     3     3     .     1     3     .     .     .     .     .      .      .
  15     3     .     3     .     .     .     .     1     .     .      .      2
  16     .     1     1     3     .     3     .     .     .     .      .      .
  17     2     .     .     .     .     2     .     .     .     3      3      .
  18     .     .     .     .     .     .     1     3     2     3      .      .
  19     .     .     .     .     3     1     .     1     .     .      .      1
  20     1     2     .     .     .     .     .     .     2     .      3      .
  21     .     .     .     3     1     .     .     1     .     .      2      .
  22     .     .     .     .     1     3     2     .     .     .      .      1
  23     3     .     .     .     .     1     3     .     .     1      .      .
  24     .     .     1     .     1     .     .     2     .     .      .      2
  25     .     1     .     .     .     .     2     2     .     1      .      .
  26     .     3     .     .     .     .     3     1     .     2      .      .
  27     .     .     .     1     .     1     2     .     .     2      .      .
  28     .     1     .     .     .     .     3     .     .     .      2      2
  29     2     .     2     .     2     .     .     .     2     .      .      .
  30     .     .     .     3     .     .     .     .     1     1      .      2
  31     .     .     .     2     .     2     .     .     2     .      1      .
  32     3     .     .     3     .     .     .     2     .     .      .      1
  33     .     2     .     .     2     2     .     3     .     .      .      .
  34     .     3     3     .     .     1     .     .     .     .      2      .
  35     .     .     .     1     .     3     .     2     1     .      .      .
  36     1     .     .     2     .     .     .     .     3     .      .      3
  37     .     .     2     2     .     .     .     3     .     .      3      .
```

Notice that the first run is constant. For all other runs, exactly four factors vary and have levels not missing.

Choice from Triples; Partial Profiles Constructed Using Restrictions

The approach we just saw, constructing partial profiles using the `partial=` option, is fine for a full-profile conjoint study or a pairwise choice study with level shifts. However, it is not good for a more general choice experiment with more alternatives. For a choice experiment, you need partial-profile restrictions on each alternative, and you must have the same attributes varying in every alternative within each choice set. There is currently no automatic way to request this in the `%MktEx` macro, so you have to program the restrictions yourself. To specify restrictions for choice designs, you need to take into consideration the number of attributes that can vary within each alternative, which ones, and which attributes go with which alternatives. Fortunately, that is not too difficult. See page 471 for another example of restrictions.

In this section, we construct a partial-profile design for a purely generic (unbranded) study, with ten attributes and three alternatives. Each attribute has three levels, and each alternative is a bundle of attributes. Partial-profile designs have the advantage that subjects do not have to consider all attributes at once. However, this is also a bit of a disadvantage as well in the sense that the subjects must constantly shift from considering one set of attributes to considering a different set. For this reason, it can be helpful to get more information out of each choice, and having more than two alternatives per choice set accomplishes this.

This example has several parts. As we mentioned in the chair study, we usually do not directly use the `%MktEx` macro to generate designs for generic studies. Instead, we use the `%MktEx` macro to generate a candidate set of partial-profile choice sets. Next, the design is checked and turned into a candidate set of choice sets. Next, the `%MktDups` macro is called to ensure there are no duplicate choice sets. Finally, the `%ChoicEff` macro is used to create an efficient generic partial-profile choice design.

Before we go into any more detail on making this design, let's skip ahead and look at a couple of potential choice sets so it is clear what we are trying to accomplish and why. Two potential choice sets, still in linear arrangement, are as follows:

```
    2 2 1 3 1 2 1 2 2 2    2 1 3 2 1 1 2 1 2 2    2 3 2 1 1 3 3 3 2 2
    2 2 1 3 2 2 1 3 2 3    3 2 2 3 1 2 1 1 1 1    1 2 3 3 3 2 1 2 3 2
```

The same two potential choice sets, but now arrayed in choice design format, are as follows:

```
                              Partial Profiles

           Set    x1    x2    x3    x4    x5    x6    x7    x8    x9    x10

             1     2     2     1     3     1     2     1     2     2     2
                   2     1     3     2     1     1     2     1     2     2
                   2     3     2     1     1     3     3     3     2     2

             2     2     2     1     3     2     2     1     3     2     3
                   3     2     2     3     1     2     1     1     1     1
                   1     2     3     3     3     2     1     2     3     2
```

Each choice set has 10 three-level factors and three alternatives. Four attributes are constant in each choice set: `x1`, `x5`, `x9`, and `x10` in the first choice set, and `x2`, `x4`, `x6`, and `x7` in the second choice set. We do not need an all-constant choice set like we saw in our earlier partial-profile designs, nor do we need an extra level for not varying. In this approach, we simply construct choice sets for four constant attributes (they can be constant at 1, 2, or 3) and six varying attributes (with levels: 1, 2, and 3). Respondents are given a choice task along the lines of “Given a set of products that differ on these attributes but are identical in all other respects, which one would you choose?”. They are then shown a list of differences.

The following steps create the candidate design:

```
title 'Partial Profiles';

%macro partprof;
   sum = 0;
   do k = 1 to 10;
      sum = sum + (x[k] = x[k+10]   &   x[k] = x[k+20]);
      end;
   bad = abs(sum - 4);
   %mend;

%mktex(3 ** 30,                    /* 30 three-level factors              */
       n=198,                      /* 198 runs                            */
       options=quickr              /* very quick run with random init     */
               nox,                /* suppress x1, x2, ... creation       */
       order=random,               /* loop over columns in a random order */
       out=sasuser.cand,           /* output design stored permanently    */
       restrictions=partprof,      /* name of restrictions macro          */
       seed=382)                   /* random number seed                  */
```

We requested a design in 198 runs with 30 three-level factors. The 198 is chosen arbitrarily as a number divisible by $3 \times 3 = 9$ that gives us approximately 200 candidate sets. The first ten factors, `x1-x10`, make the first alternative, the next ten, `x11-x20`, make the second alternative, and the last ten, `x21-x30`, make the third alternative. We want six attributes to be nonconstant at a time. The `PartProf` macro counts the number of constant attributes: `x1 = x11 = x21`, `x2 = x12 = x22`, ..., and `x10 = x20 = x30`. If the number of constant attributes is four, our choice set conforms. If it is more or less than four, our choice set is in violation of the restrictions. The badness is the absolute difference between four and the number of constant attributes.

We specify `order=random` so the columns are looped over in a random order in the coordinate-exchange algorithm. When `partial=` is specified, as it is in the previous partial-profile examples, `order=random` is the default. Whenever you are imposing partial-profile restrictions without using the `partial=` option, you should specify `order=random`. Without `order=random`, `%MktEx` tends to put the nonconstant attributes close together in each row.

Our goal in this step is to make a candidate set of potential partial-profile choice sets, not to make a final experimental design. Ideally, it is nice to have more than random candidates—it is nice if our candidate generation code at least makes some attempt to ensure that our attributes are approximately orthogonal and balanced across attributes both between and within alternatives. This is a big problem (30 factors and 198 runs) with restrictions, so the `%MktEx` macro runs slowly by default. It is not critical that we permit the macro to spend a great deal of time optimizing linear model *D*-efficiency.

For this reason, we limit the number of iterations by specifying `options=quickr`. This is equivalent to specifying `optiter=0`, which specifies no OPTEX iterations, since with large partial-profile studies, we will never have a good candidate set for PROC OPTEX to search. It also specifies `tabiter=0` since an orthogonal array initial design is horrible for this problem. We also specified `options=nox` to increase efficiency by suppressing the creation of `x1`, `x2`, and so on, since they are not needed, and our restrictions are just based on `x`. Some of the results are as follows:

```
                           Partial Profiles

                       Algorithm Search History

                              Current         Best
      Design    Row,Col  D-Efficiency  D-Efficiency  Notes
      ----------------------------------------------------------
           1      Start       85.1531                Ran,Mut,Ann
           1    169   1       93.3863       93.3863  Conforms
           1    169   8       93.3892       93.3892
           1    169  22       93.3892       93.3892
           1    169  12       93.3911       93.3911
           1    170  20       93.3954       93.3954
           .
           .
           .
           1    123   1       96.5805       96.5805
           1     38  21       96.5806       96.5806
           1     47   1       96.5811       96.5811
           1        End       96.5811
           .
           .
           .
                           Partial Profiles

                         The OPTEX Procedure

                                                                  Average
                                                                Prediction
 Design                                                          Standard
 Number     D-Efficiency     A-Efficiency     G-Efficiency        Error
 ------------------------------------------------------------------------
      1          96.5811          93.5359          95.3976         0.5551
```

The macro finds a design that conforms to the restrictions (shown by the `Conforms` note). This step took approximately 3 minutes.

The rest of the code for making the partial-profile choice design is as follows:

```
%mktkey(3 10)

%mktroll(design=sasuser.cand, key=key, out=rolled)

%mktdups(generic, data=rolled, out=nodups, factors=x1-x10, nalts=3)

proc print data=nodups(obs=9); id set; by set; run;

%choiceff(data=nodups,              /* candidate set of choice sets        */
          model=class(x1-x10 / sta),/* model with stdzd orthogonal coding  */
          seed=495,                 /* random number seed                  */
          maxiter=10,               /* maximum iterations for each phase   */
          nsets=27,                 /* number of choice sets               */
          nalts=3,                  /* number of alternatives              */
          options=nodups            /* no duplicate choice sets            */
                  relative,         /* display relative D-efficiency       */
          beta=zero)                /* assumed beta vector, Ho: b=0        */

proc print data=best; id set; by notsorted set; var x1-x10; run;
```

The `%MktKey` macro is run to generate a `Key` data set with 3 rows, 10 columns and the variable names `x1 - x30`. The `Key` data set is as follows:

```
                              Partial Profiles

          x1     x2     x3     x4     x5     x6     x7     x8     x9     x10

          x1     x2     x3     x4     x5     x6     x7     x8     x9     x10
          x11    x12    x13    x14    x15    x16    x17    x18    x19    x20
          x21    x22    x23    x24    x25    x26    x27    x28    x29    x30
```

The `%MktRoll` macro creates a choice design from the linear candidate design.

The next step uses the `%MktDups` macro to check a design to see if there are any duplicate runs and output just the unique ones. For a generic study like this, it can also check to make sure there are duplicate choice sets taking into account the fact that two choice sets can be duplicates even if the alternatives are not in the same order. The `%MktDups` step names in a positional parameter the type of design as a `generic` choice design. It names the input data set and the output data set that contain the design with any duplicates removed. It names the factors in the choice design `x1-x10` and the number of alternatives. The result is a data set called `NoDups`. The first 3 candidate choice sets are as follows:

```
                              Partial Profiles

     Set     _Alt_    x1    x2    x3    x4    x5    x6    x7    x8    x9    x10

      1        1       1     1     1     1     1     3     1     1     2     1
               2       2     3     2     3     3     2     1     1     2     1
               3       2     2     3     2     2     1     1     1     2     1

      2        1       1     1     1     1     2     3     3     2     2     2
               2       2     3     1     1     2     2     1     3     3     2
               3       3     2     1     1     2     3     2     1     1     2

      3        1       1     1     1     1     2     3     3     3     1     3
               2       3     1     1     1     2     2     3     1     3     2
               3       2     1     2     1     2     3     3     2     2     1
```

The `%ChoicEff` macro is called to search for an efficient choice design. The model specification `class(x1-x10 / sta)` specifies a generic model with 10 attributes and the standardized orthogonal contrast coding. The option `maxiter=10` specifies more than the default number of iterations (the default is 2 designs). We ask for a design with 27 sets and 3 alternatives. Furthermore, we ask for no duplicate choice sets, relative *D*-efficiency, and specify an assumed beta vector of zero. Some of the results from the `%ChoicEff` macro are as follows:

```
                              Partial Profiles

                   Design   Iteration  D-Efficiency       D-Error
                   ----------------------------------------------
                        1       0         12.57808 *      0.07950
                                1         14.91821 *      0.06703
                                2         14.92197 *      0.06702

                      .
                      .
                      .

                   Design   Iteration  D-Efficiency       D-Error
                   ----------------------------------------------
                       10       0         12.55237        0.07967
                                1         14.95788        0.06685
                                2         15.05607        0.06642
                                3         15.06856        0.06636
```

```
                          Partial Profiles

                           Final Results

                     Design                   9
                     Choice Sets             27
                     Alternatives             3
                     Parameters              20
                     Maximum Parameters      54
                     D-Efficiency       15.3187
                     Relative D-Eff     56.7360
                     D-Error             0.0653
                     1 / Choice Sets     0.0370

                          Partial Profiles

             Variable                                       Standard
        n      Name       Label     Variance    DF       Error

        1      x11        x1 1      0.069368     1     0.26338
        2      x12        x1 2      0.061410     1     0.24781
        3      x21        x2 1      0.069033     1     0.26274
        4      x22        x2 2      0.065628     1     0.25618
        5      x31        x3 1      0.072568     1     0.26938
        6      x32        x3 2      0.082373     1     0.28701
        7      x41        x4 1      0.071943     1     0.26822
        8      x42        x4 2      0.074494     1     0.27294
        9      x51        x5 1      0.068258     1     0.26126
       10      x52        x5 2      0.070409     1     0.26535
       11      x61        x6 1      0.063116     1     0.25123
       12      x62        x6 2      0.065635     1     0.25619
       13      x71        x7 1      0.062414     1     0.24983
       14      x72        x7 2      0.069006     1     0.26269
       15      x81        x8 1      0.071232     1     0.26689
       16      x82        x8 2      0.080080     1     0.28298
       17      x91        x9 1      0.062817     1     0.25063
       18      x92        x9 2      0.068655     1     0.26202
       19      x101       x10 1     0.063551     1     0.25209
       20      x102       x10 2     0.069304     1     0.26326
                                                ==
                                                20
```

Part of the design is as follows:

```
                           Partial Profiles

 Set    x1    x2    x3    x4    x5    x6    x7    x8    x9    x10

 197     3     3     3     3     1     3     3     3     2     3
         3     1     3     2     1     1     2     2     3     3
         3     2     3     1     1     2     1     1     1     3

 191     3     3     3     1     3     1     2     1     3     2
         3     1     1     2     3     1     3     1     1     1
         3     2     2     3     3     1     1     1     2     3
   .
   .
   .

 110     2     3     1     1     1     3     1     3     1     2
         3     3     3     3     1     3     3     1     1     1
         1     3     2     2     1     3     3     2     1     3
```

The design has 27 choice sets. The choice set numbers shown in this output correspond to the *original* set numbers in the candidate design not the choice set numbers in the final design. This design has a relative *D*-efficiency of 56.7360. It can be no larger than 60 since 6 of 10 attributes vary.

Six Alternatives; Partial Profiles Constructed Using Restrictions

In this next example, we construct a partial-profile design with 20 binary attributes and six alternatives with 15 attributes fixed at the base-line level of 1 for each alternative. Our partial-profile restriction macro is an obvious modification of the one used in the previous example. Our linear arrangement needs $6 \times 20 = 120$ factors. The first attribute is made from `x1`, `x21`, `x41`, `x61`, `x81`, and `x101`; the second attribute is made from `x2`, `x22`, `x42`, `x62`, `x82`, and `x102`; and so on. The `do` loop counts the number of times all of the linear factors within an attribute are equal to one and our badness function increases as the number of constant attributes deviates from 15. The following step creates the macro:

```
%macro partprof;
   sum = 0;
   do k = 1 to 20;
      sum = sum + (x[k]    = 1 & x[k+20] = 1 & x[k+40]  = 1 &
                   x[k+60] = 1 & x[k+80] = 1 & x[k+100] = 1);
      end;
   bad = abs(sum - 15);
   %mend;
```

The `%MktEx` macro is run as follows:

```
%mktex(2 ** 120,                   /* 120 two-level factors                */
       n=300,                      /* 300 runs                             */
       order=random,               /* loop over columns in a random order  */
       out=cand,                   /* output design                        */
       restrictions=partprof,      /* name of restrictions macro           */
       seed=424,                   /* random number seed                   */
       maxtime=0,                  /* stop once design conforms            */
       options=largedesign         /* make large design more quickly       */
               nosort              /* do not sort output design            */
               resrep              /* detailed report on restrictions      */
               quickr              /* very quick run with random init      */
               nox)                /* suppress x1, x2, ... creation        */
```

This step requests 120 binary factors and 300 runs. We specify `order=random` so that the columns are looped over in a random order in the coordinate-exchange algorithm. You should always specify `order=random` with partial-profile designs. With a sequential order you tend to get nonvarying attributes paired with only nearby attributes. Several options are specified to make this step run quickly. With `maxtime=0`, only one design is created. With `options=quickr`, that one design is created with the coordinate exchange algorithm and a random initialization. (The “r” in `quickr` stands for random.) The `largedesign` option allows `%MktEx` to stop as soon as it has imposed all restrictions. The `resrep` option reports on the progress of imposing restrictions.

The first part of the output is as follows:

```
                          Algorithm Search History

                                   Current         Best
           Design    Row,Col  D-Efficiency  D-Efficiency  Notes
           ----------------------------------------------------------
                1      Start       79.5699                Ran,Mut,Ann
                1      1           79.8384                14 Violations
                1      2           80.0541                12 Violations
                1      3           80.3047                14 Violations
                1      4           80.5649                15 Violations
                1      5           80.7840                13 Violations
                1      6           81.0383                12 Violations
                .
                .
                .
```

At the end of the first pass through the design we see the following warning:

```
WARNING: It may be impossible to meet all restrictions.
```

The macro never gets anywhere in imposing restrictions. Every row of the design has many restriction violations as shown by the output from the `resrep` (restrictions report) option. Always specify `options=resrep` with complicated restrictions so you can look for things like this. When the macro is never succeeding in imposing restrictions, there is something wrong with the restrictions macro. The macro we used is as follows:

```
%macro partprof;
   sum = 0;
   do k = 1 to 20;
      sum = sum + (x[k]    = 1 & x[k+20] = 1 & x[k+40]  = 1 &
                   x[k+60] = 1 & x[k+80] = 1 & x[k+100] = 1);
      end;
   bad = abs(sum - 15);
   %mend;
```

It correctly evaluates the Boolean expression to determine whether an attribute is all one, and it correctly counts the number of such attributes. It also correctly sets `bad` to the absolute difference between the sum and 15, the desired number of constant attributes. Sometimes you get results like we just saw when you made a logical mistake in programming the restrictions. For example, you might have written a set of restrictions that are impossible to satisfy. That is not the problem in this case. The problem in this case is the quantification of badness is not fine enough. You need to tell the macro whenever it does something that moves it closer to an acceptable solution. Consider a potential choice set that almost conforms to the restrictions such as the following:

```
Set x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17 x18 x19 x20

 1   2  1  1  1  2  2  1  1  1  1   1   1   2   1   1   1   1   1   1   1
     2  1  2  1  2  2  1  1  1  1   1   1   2   1   1   1   1   1   1   1
     1  1  2  1  1  2  2  1  1  1   1   1   2   1   1   1   1   1   1   1
     1  1  2  1  2  2  1  1  1  1   1   1   1   1   1   1   1   1   1   1
     2  1  1  1  1  1  2  1  1  1   1   1   1   1   1   1   1   1   1   1
     1  1  1  1  1  1  1  1  1  1   1   1   1   1   1   1   1   1   1   1
```

The problem with this choice set is it has six nonconstant attributes instead of five. So `bad = abs(14 - 15) = 1`. Now consider what happens when the macro changes the first attribute of the first alternative from 2 to 1. This is a change in the right direction because it moves the choice set closer to having one more constant attribute. However, this change has no effect on the badness criterion. It is still 1, because we still have six nonconstant attributes. We are not giving `%MktEx` enough information about when it is heading in the right direction. The `%MktEx` macro, without your restrictions macro to otherwise guide it, is guided by the goal of maximizing *D*-efficiency. It does not particularly want to make partial-profile designs, because imposing all of those ties decreases statistical *D*-efficiency. Using the hill climbing analogy, `%MktEx` wants to climb Mount Everest; your restrictions macro needs to tell it to find the top of an island in the middle of a river valley. This is not where `%MktEx` would normally look. If your restrictions macro is going to overcome the `%MktEx` macro's normal goal, you have to train it and more explicitly tell it where to look.

Training the `%MktEx` macro to find highly restricted designs is like training a dog. You have to be persistent and consistent, and you need to watch it every second. You have to reward it whenever it does the right thing and you have to punish it for even thinking about doing the wrong thing. When it tears up your favorite slippers, you need to whack it over the head with a rolled up newspaper.[*] It

[*]No actual dogs were harmed in the process of developing any of this software.

is eager to please, but it is easily tempted, and it is not smart enough to figure out what to do unless you very explicitly tell it. It will run wherever it wants unless you keep it on a short leash. With that in mind, the revised partial-profile restrictions macro is as follows:

```
%macro partprof;
   sum = 0;
   do k = 1 to 20;
      sum = sum + (x[k]    = 1 & x[k+20] = 1 & x[k+40]  = 1 &
                   x[k+60] = 1 & x[k+80] = 1 & x[k+100] = 1);
      end;
   bad = abs(sum - 15);
   if sum < 15 & x[j1] = 2 then do;
      k = mod(j1 - 1, 20) + 1;
      c = (x[k]    = 1) + (x[k+20] = 1) + (x[k+40]  = 1) +
          (x[k+60] = 1) + (x[k+80] = 1) + (x[k+100] = 1);
      if c >= 3 then bad = bad + (6 - c) / 6;
      end;
   %mend;
```

It starts the same way as the old one, but it has some additional fine tuning. Like before, this macro counts the number of times that all six alternatives equal 1 and stores the result in `sum`. When `sum` is less than 15, we need more constant attributes. When the current level of the current factor, `x[j1]`, is 2, the macro next considers whether it should change the 2 to a 1. First, we compute `k`, the attribute number and evaluate the number of ones in that attribute, and store that in `c`. (For example, when `j1`, the linear factor number, equals 2, 22, 42, 62, 82, or 102, we are working with attribute `k` = 2.) If it looks like this factor is going to be constant (three or more ones), we add the proportion of twos to the badness function (more twos is worse).

Consider again the first row of the sample choice set shown previously. Badness starts out as 1 since `sum` is 14. Since badness is nonzero and since we are looking at a 2 in the first column, badness is increased by 1/2, which is the proportion of twos. Now consider changing that first two to a one. Now badness is 1 + 1/3, which is smaller than 1 + 1/2 so changing 2 to 1 moves badness in the right direction.

There are many other ways that you could write a restrictions macro for this problem. However you do it, you need to provide a quantification of badness that guides the macro toward acceptable designs. Most of the time, for complicated restrictions, you will not be able to sit down and write a good restrictions macro off of the top of your head.* It requires some trial and error to get something that works. For this reason, at least at first, you should specify `maxtime=0, options=largedesign resrep quickr` so you can see if the macro is succeeding in imposing restrictions, and so the macro stops quickly when it has succeeded. Then you need to carefully check your design. It is not unusual for the macro to succeed splendidly only to find you gave it the wrong set of restrictions. We can run the `%MktEx` macro exactly as before to test our new macro:

*This one might look simple once you see it, but it took me several tries to get it right. I had previous versions that worked quite well in spite of some logical flaws that caused them to quantify badness in not quite the way that I intended.

```
%mktex(2 ** 120,                 /* 120 two-level factors               */
       n=300,                    /* 300 runs                            */
       order=random,             /* loop over columns in a random order */
       out=cand,                 /* output design                       */
       restrictions=partprof,    /* name of restrictions macro          */
       seed=424,                 /* random number seed                  */
       maxtime=0,                /* stop once design conforms           */
       options=largedesign       /* make large design more quickly      */
               nosort            /* do not sort output design           */
               resrep            /* detailed report on restrictions     */
               quickr            /* very quick run with random init     */
               nox)              /* suppress x1, x2, ... creation       */
```

Some of the output from the first pass through the design is as follows:

```
                           Algorithm Search History

                              Current          Best
         Design    Row,Col  D-Efficiency  D-Efficiency  Notes
         ----------------------------------------------------------
              1      Start       79.5699                Ran,Mut,Ann
              1       1          79.5426                0 Violations
              1       2          79.5904                1 Violations
              1       3          79.6679                4 Violations
              1       4          79.6601                2 Violations
              1       5          79.6136                0 Violations
              .
              .
              .
              1     100          70.7815                0 Violations
              1     101          70.6383                3.5 Violations
              1     102          70.4911                0 Violations
              .
              .
              .
              1     197          54.9906                2.5 Violations
              1     198          54.8216                2 Violations
              1     199          54.6956                3 Violations
              1     200          54.5982                4 Violations
              1     201          54.4009                3 Violations
              1     202          54.2690                5 Violations
              1     203          54.0316                1 Violations
              .
              .
              .
              1     297          37.1307                4 Violations
              1     298          36.9190                0 Violations
              1     299          36.7720                4 Violations
              1     300          36.5326                2 Violations
```

This iteration history looks better. At least some choice sets conform. Most do not however. Notice the fractional number of violations in some rows. Throughout most of this iteration history, *D*-efficiency is steadily going down as more and more restrictions are imposed. Part of the iteration history for the second pass through the design is as follows:

```
                           Algorithm Search History

                                 Current         Best
              Design    Row,Col  D-Efficiency  D-Efficiency  Notes
              ----------------------------------------------------------
                   1      1          36.5626                 0 Violations
                   1      2          36.6017                 0 Violations
                   1      3          36.5544                 0 Violations
                   1      4          36.5408                 0 Violations
                   1      5          36.5872                 0 Violations
                   .
                   .
                   .
                   1     71          35.3957                 0 Violations
                   1     72          35.2990                 1 Violations
                   1     72          35.3149                 0 Violations
                   .
                   .
                   .
                   1    200          32.7482                 0 Violations
                   1    201          32.7255                 0 Violations
                   1    202          32.6120                 0 Violations
                   .
                   .
                   .
                   1    298          30.8923                 0 Violations
                   1    299          30.8077                 0 Violations
                   1    300          30.7819                 0 Violations
```

This part of the iteration history looks much better. The table contains a number of rows like we see in row 72. The `%MktEx` macro attempts to impose restrictions and it does not quite succeed, so it immediately tries again, one or more times, until it succeeds or gives up. In this case, it always succeeds. *D*-efficiency is still mostly decreasing. Note that the “0 Violations” that we see in this table tells us that there are no violations remaining in the indicated row. There might very well still be violations in other rows. However by the end of this pass, the restrictions are almost certainly completely imposed, so we should see *D*-efficiency start back up. Some of the output from the next pass through the design is as follows:

```
                        Algorithm Search History

                                  Current         Best
Design    Row,Col  D-Efficiency  D-Efficiency  Notes
----------------------------------------------------------
     1      1           30.7883                0 Violations
     1      2           30.8000                0 Violations
     1      3           30.8232                0 Violations
     .
     .
     .
     1    100           32.2841                0 Violations
     1    101           32.3019                0 Violations
     1    102           32.3076                0 Violations
     .
     .
     .
     1    200           33.1653                0 Violations
     1    201           33.1732                0 Violations
     1    202           33.1784                0 Violations
     .
     .
     .
     1    298           33.8588                0 Violations
     1    299           33.8877                0 Violations
     1    300           33.8971                0 Violations
```

Now *D*-efficiency is increasing. The rest of the iteration history is as follows:

```
                        Algorithm Search History

                                  Current         Best
Design    Row,Col  D-Efficiency  D-Efficiency  Notes
----------------------------------------------------------
     1      1   1       33.8971       33.8971  Conforms
     1      1  53       33.8988       33.8988
     1      1  60       33.9013       33.9013
     1      1   6       33.9016       33.9016
     1        End       33.9016
```

It is followed by the following messages:

```
NOTE: Stopping early, possibly before convergence, with a large design.
```

Due to `maxtime=0, options=largedesign quickr`, the macro stops after it has completed one pass through the design without encountering any restriction violations. Our final *D*-efficiency = 33.9016 is not very high, but this is just a candidate set. Furthermore, this is a *highly* restricted design, so it is not surprising that *D*-efficiency is not very high. We might want to iterate more and see if we can do better, but for now, let's test the rest of our code. The `%MktEx` step took about 2.5 minutes. When the macro completes a full pass through the design without detecting any violations, it displays `Conforms` and switches from the `options=resrep` style to the normal iteration history style. The following steps turn the linear arrangement into a choice design and eliminate duplicate choice sets:

```
%mktkey(6 20)

%mktroll(design=cand, key=key, out=rolled)

proc print; by set; id set; var x:; where set le 5; run;

%mktdups(generic, data=rolled, out=nodups, factors=x1-x20, nalts=6)

proc print data=nodups(obs=18); id set; by set; run;
```

The `%MktKey` macro is run to generate a `Key` data set with 6 rows, 20 columns and the variable names `x1 - x120`. The `Key` data set, displayed in two panels, is as follows:

```
 x1      x2      x3      x4      x5      x6      x7      x8      x9      x10

x1      x2      x3      x4      x5      x6      x7      x8      x9      x10
x21     x22     x23     x24     x25     x26     x27     x28     x29     x30
x41     x42     x43     x44     x45     x46     x47     x48     x49     x50
x61     x62     x63     x64     x65     x66     x67     x68     x69     x70
x81     x82     x83     x84     x85     x86     x87     x88     x89     x90
x101    x102    x103    x104    x105    x106    x107    x108    x109    x110

x11     x12     x13     x14     x15     x16     x17     x18     x19     x20

x11     x12     x13     x14     x15     x16     x17     x18     x19     x20
x31     x32     x33     x34     x35     x36     x37     x38     x39     x40
x51     x52     x53     x54     x55     x56     x57     x58     x59     x60
x71     x72     x73     x74     x75     x76     x77     x78     x79     x80
x91     x92     x93     x94     x95     x96     x97     x98     x99     x100
x111    x112    x113    x114    x115    x116    x117    x118    x119    x120
```

The `%MktDups` macro eliminated 70 choice sets with duplicate alternatives resulting in a candidate set with 230 choice sets.

The results of the last PROC PRINT, with the first three candidate choice sets, are as follows:

```
Set x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17 x18 x19 x20

 1   1  1  1  1  2  1  1  1  1  1   1   1   1   1   1   1   1   1   1   2
     1  1  1  1  1  1  1  1  2  1   1   1   2   1   1   1   1   1   1   1
     1  1  1  1  2  1  1  1  2  1   1   1   1   1   1   1   1   1   1   1
     1  1  1  1  1  2  1  1  1  1   1   1   2   1   1   1   1   1   1   2
     1  1  1  1  2  2  1  1  2  1   1   1   1   1   1   1   1   1   1   2
     1  1  1  1  1  1  1  1  1  1   1   1   2   1   1   1   1   1   1   1

 2   1  2  1  1  1  1  1  1  1  1   1   1   2   1   1   2   2   1   2   1
     1  2  1  1  1  1  1  1  1  1   1   1   1   1   1   2   2   1   2   1
     1  2  1  1  1  1  1  1  1  1   1   1   1   1   1   2   2   1   2   1
     1  1  1  1  1  1  1  1  1  1   1   1   2   1   1   2   1   1   1   1
     1  1  1  1  1  1  1  1  1  1   1   1   2   1   1   1   1   1   2   1
     1  1  1  1  1  1  1  1  1  1   1   1   1   1   1   1   1   1   1   1

 3   2  1  1  1  1  1  1  2  2  1   1   1   1   2   1   1   1   1   1   1
     1  1  1  1  1  1  1  2  1  1   1   1   1   1   1   2   1   1   1   1
     2  1  1  1  1  1  1  1  2  1   1   1   1   2   1   2   1   1   1   1
     1  1  1  1  1  1  1  1  1  1   1   1   1   1   1   2   1   1   1   1
     1  1  1  1  1  1  1  2  2  1   1   1   1   1   1   2   1   1   1   1
     2  1  1  1  1  1  1  2  2  1   1   1   1   2   1   1   1   1   1   1
```

Each choice set has the correct number of nonconstant attributes. The following step runs the `%ChoicEff` macro to find a choice design:

```
%choiceff(data=nodups,              /* candidate set of choice sets         */
          model=class(x1-x20 / zero=first), /* main effects, first ref level*/
          seed=495,                 /* random number seed                   */
          maxiter=10,               /* maximum iterations for each phase    */
          nsets=18,                 /* number of choice sets                */
          nalts=6,                  /* number of alternatives               */
          options=nodups,           /* no duplicate choice sets             */
          beta=zero)                /* assumed beta vector, Ho: b=0         */

proc print data=best; id set; by notsorted set; var x1-x20; run;
```

We set the reference level to the first level of each factor, which is the base-line level of 1. Some of the output is as follows:

```
                    n    Name    Beta    Label

                    1    x12       0     x1 2
                    2    x22       0     x2 2
                    3    x32       0     x3 2
                    4    x42       0     x4 2
                    5    x52       0     x5 2
                    6    x62       0     x6 2
                    7    x72       0     x7 2
                    8    x82       0     x8 2
                    9    x92       0     x9 2
                   10    x102      0     x10 2
                   11    x112      0     x11 2
                   12    x122      0     x12 2
                   13    x132      0     x13 2
                   14    x142      0     x14 2
                   15    x152      0     x15 2
                   16    x162      0     x16 2
                   17    x172      0     x17 2
                   18    x182      0     x18 2
                   19    x192      0     x19 2
                   20    x202      0     x20 2

         Design   Iteration  D-Efficiency       D-Error
         ----------------------------------------------
              1       0            0                 .
                      1          1.02522 *     0.97540
                      2          1.03841 *     0.96301
                      3          1.03875 *     0.96269

         Design   Iteration  D-Efficiency       D-Error
         ----------------------------------------------
              2       0          0.82624       1.21030
                      1          1.02442       0.97616
                      2          1.03920 *     0.96228
                      3          1.04403 *     0.95783

             .
             .
             .

         Design   Iteration  D-Efficiency       D-Error
         ----------------------------------------------
             10       0          0.84581       1.18229
                      1          1.01501       0.98521
                      2          1.03206       0.96893
                      3          1.04232       0.95940
                      4          1.04232       0.95940
```

```
                        Final Results

                  Design                    2
                  Choice Sets              18
                  Alternatives              6
                  Parameters               20
                  Maximum Parameters       90
                  D-Efficiency         1.0501
                  D-Error              0.9495

            Variable                                      Standard
       n      Name       Label      Variance     DF        Error

       1      x12        x1 2        0.97668      1      0.98827
       2      x22        x2 2        1.10960      1      1.05338
       3      x32        x3 2        1.10506      1      1.05122
       4      x42        x4 2        0.87763      1      0.93682
       5      x52        x5 2        0.92710      1      0.96286
       6      x62        x6 2        0.75023      1      0.86616
       7      x72        x7 2        0.92874      1      0.96371
       8      x82        x8 2        1.10497      1      1.05118
       9      x92        x9 2        1.06918      1      1.03401
      10      x102       x10 2       1.08282      1      1.04058
      11      x112       x11 2       1.14818      1      1.07153
      12      x122       x12 2       0.85384      1      0.92404
      13      x132       x13 2       0.84918      1      0.92151
      14      x142       x14 2       1.10046      1      1.04903
      15      x152       x15 2       1.09171      1      1.04485
      16      x162       x16 2       0.87602      1      0.93596
      17      x172       x17 2       1.05549      1      1.02737
      18      x182       x18 2       0.86694      1      0.93109
      19      x192       x19 2       1.06911      1      1.03398
      20      x202       x20 2       1.08380      1      1.04106
                                                 ==
                                                 20


Set x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17 x18 x19 x20

227  1  1  1  1  1  1  1  1  1  1   2   1   1   1   1   1   1   1   1   1
     1  1  2  1  1  1  1  1  1  1   1   1   1   1   1   1   1   1   1   2
     1  1  2  1  1  1  1  1  1  1   2   1   1   1   1   2   1   1   1   1
     1  1  1  1  1  1  1  1  1  1   1   1   2   1   1   2   1   1   1   2
     1  1  1  1  1  1  1  1  1  1   1   1   2   1   1   1   1   1   1   1
     1  1  2  1  1  1  1  1  1  1   1   1   2   1   1   2   1   1   1   2
```

```
187  1  1  1  1  1  1  1  1  1  2   1   2   1   1   1   1   1   2   1   1
     1  1  1  2  1  1  1  1  1  1   1   2   1   1   2   1   1   1   1   1
     1  1  1  2  1  1  1  1  1  2   1   1   1   1   2   1   1   2   1   1
     1  1  1  2  1  1  1  1  1  2   1   2   1   1   1   1   1   1   1   1
     1  1  1  2  1  1  1  1  1  1   1   1   1   1   2   1   1   2   1   1
     1  1  1  1  1  1  1  1  1  1   1   1   1   1   1   1   1   1   1   1
  .
  .
  .

191  1  1  2  1  2  1  1  1  1  1   1   1   1   1   1   1   1   1   1   1
     1  1  2  1  1  1  1  1  1  1   1   1   1   1   1   1   1   2   1   1
     1  1  2  1  1  1  1  2  1  1   1   1   1   2   1   1   1   1   1   1
     1  1  1  1  1  1  1  2  1  1   1   1   1   2   1   1   1   1   1   1
     1  1  1  1  2  1  1  2  1  1   1   1   1   1   1   1   1   2   1   1
     1  1  1  1  2  1  1  1  1  1   1   1   1   2   1   1   1   2   1   1
```

Since the `%ChoicEff` macro did not have any problems, and the results look reasonable, it appears that we did everything right. Now we could try again, perhaps with more choice sets (say 400), and we could iterate longer (say one hour), then we could make our final partial-profile choice design. This is illustrated in the following step:

```
%mktex(2 ** 120,                   /* 120 two-level factors              */
       n=400,                      /* 400 runs                           */
       optiter=0,                  /* no PROC OPTEX iterations           */
       tabiter=0,                  /* no OA initialization iterations    */
       maxtime=60,                 /* at most 60 minutes per phase       */
       order=random,               /* loop over columns in a random order */
       out=cand,                   /* output design                      */
       restrictions=partprof,      /* name of restrictions macro         */
       seed=424,                   /* random number seed                 */
       maxstages=1,                /* maximum number of algorithm stages  */
       options=largedesign         /* make large design more quickly     */
               nosort)             /* do not sort output design          */
```

The results of this step are not shown.

To recap, the first restrictions macro correctly differentiated between acceptable and unacceptable choice sets, but it provided `%MktEx` with no guidance or direction on how to find acceptable choice sets. Hence, the first macro did not work. The second macro corrected this problem by "nudging" `%MktEx` in the right direction. The restrictions macro looked for attributes that appear to be heading toward constant and created a penalty function that encouraged `%MktEx` to make those attributes constant. Next, we will look at another way of writing a restrictions macro for this problem. There is nothing subtle about this next approach. This next macro uses the whack-it-over-the-head-with-a-rolled-up-newspaper approach. Sometimes you need to tell `%MktEx` that a restriction is really important by strongly eliminating that source of badness. In this case, our macro strongly eliminates twos from the design until the restriction violations go away. This is illustrated in the following steps:

```
%macro partprof;
   sum = 0;
   do k = 1 to 20;
      sum = sum + (x[k]    = 1 & x[k+20] = 1 & x[k+40]  = 1 &
                   x[k+60] = 1 & x[k+80] = 1 & x[k+100] = 1);
      end;
   bad = abs(sum - 15);
   if sum < 15 & x[j1] = 2 then bad = bad + 1000;
   %mend;

%mktex(2 ** 120,                  /* 120 two-level factors              */
       n=300,                     /* 300 runs                           */
       order=random,              /* loop over columns in a random order */
       out=cand,                  /* output design                      */
       restrictions=partprof,     /* name of restrictions macro         */
       seed=424,                  /* random number seed                 */
       maxtime=0,                 /* stop once design conforms          */
       options=largedesign        /* make large design more quickly     */
               nosort             /* do not sort output design          */
               resrep             /* detailed report on restrictions    */
               quickr             /* very quick run with random init    */
               nox)               /* suppress x1, x2, ... creation      */
```

Recall that we specified `order=random` so within each choice set, the columns are traversed in a different random order. As long as there are violations, within each choice set, this macro turns random twos into ones until there are so few twos left that the violations go away. Then once all of the violations go away, it allows twos to be changed back to ones to increase *D*-efficiency.

Part of the iteration history is as follows:

```
                           Algorithm Search History

                              Current         Best
Design    Row,Col  D-Efficiency  D-Efficiency  Notes
----------------------------------------------------------
     1      Start       79.5699                Ran,Mut,Ann
     1      1           79.5701                0 Violations
     1      2           79.5231                0 Violations
     1      3           79.4445                0 Violations
     1      4           79.3049                0 Violations
     1      5           79.1400                0 Violations
     .
     .
     .
     1    100           57.7096                0 Violations
     1    101           57.5138                0 Violations
     1    102           57.2500                0 Violations
```

```
.
.
.
1    200           34.9873                 0 Violations
1    201           34.7893                 0 Violations
1    202           34.5983                 0 Violations
.
.
.
1    298           18.0811                 0 Violations
1    299           17.8622                 0 Violations
1    300           17.5956                 0 Violations
1      1           17.7230                 0 Violations
1      2           17.8323                 0 Violations
1      3           17.9371                 0 Violations
.
.
.
1    100           23.7155                 0 Violations
1    101           23.7609                 0 Violations
1    102           23.8067                 0 Violations
.
.
.
1    200           28.3416                 0 Violations
1    201           28.3683                 0 Violations
1    202           28.3953                 0 Violations
.
.
.
1    298           32.0063                 0 Violations
1    299           32.0372                 0 Violations
1    300           32.0804                 0 Violations
```

```
1       1   1       32.0804       32.0804  Conforms
1       1  13       32.0873       32.0873
1       1  27       32.0950       32.0950
1       1 105       32.0981       32.0981
1       1 103       32.1020       32.1020
1       1 113       32.1032       32.1032
1       1  65       32.1096       32.1096
1       1 106       32.1116       32.1116
1       1  86       32.1159       32.1159
1       1  43       32.1163       32.1163
1       1  47       32.1193       32.1193
1       1  45       32.1232       32.1232
1       1   3       32.1294       32.1294
1       1  53       32.1295       32.1295
1       1 107       32.1344       32.1344
1       1   6       32.1366       32.1366
1       1  73       32.1372       32.1372
1         End       32.1372
```

With this approach, all violations in each row are eliminated in the first pass through the design. The macro quits after the end of the second pass, when it has completed an entire pass without encountering any restriction violations.

Call the first approach the "nudge" approach and the second approach the "whack" approach. The nudge approach starts with *D*-efficiency on the order of 80% for the random design. After one complete pass, imposing many but not all restrictions, *D*-efficiency is down around 37%. After another pass and imposing all restrictions, it is down to around 31%. Then it creeps back up to 34%. *D*-efficiency for the choice design is approximately 1.05. The whack approach starts with the same random design (due to the same random number seed) and with the same *D*-efficiency on the order of 80%. Then after one pass of severe restriction imposition, *D*-efficiency drops to 18%. The nudge approach asks "are you a good two or a bad two?" and then acts accordingly. In contrast, when the whack approach sees a two, it whacks it—no questions asked. There is no subtle nudging in the whack approach, and initially, it over corrects to impose restrictions. Hence, the design it makes at the end of the first pass is not very good, but once all of the restrictions are in place, *D*-efficiency quickly recovers to 32%. This is not quite as high as the nudge approach, but the nudge approach had one more complete pass through the design.

It is natural to ask, which approach is better? There are many ways to get `%MktEx` to impose the restrictions. It is not clear that one is better than the other. Our goal here is to use `%MktEx` to create a set of candidates for the `%ChoicEff` macro to search. Any restrictions macro that accomplishes this should be fine. Writing a macro that uses the whack approach is probably a bit easier. However, there is always some worry that the initial over correction might not be a good thing. In contrast, subtle nudging takes longer to get to the point of all restrictions being met, and you have to be a bit creative sometimes to write nudges that actually work. Sometimes you need to combine both approaches—whack it to take care of one set of restrictions then nudge it in the right direction for a secondary set of restrictions. This is all part of the art of sophisticated experimental design. Note that it is not the fact that we used a large penalty of 1000 that makes this approach an example of the whack approach. It is the whack approach because we strongly over-corrected every violation.

The differential weighting of the components of badness is another very important strategy to keep in mind. When there is more than one aspect to the set of restrictions, they sometimes trade off against each other. Every time `%MktEx` fixes one thing, something else gets worse. Differentially weighting the components, as we did in the partprof macro, can greatly help avoid this. The first component, `bad = abs(sum - 15);` sets badness to values in the range 0 to 15. The next component `if sum < 15 & x[j1] = 2 then bad = bad + 1000;` adds contributions in the thousands. `%MktEx` will never increase the second component to decrease the first. This gives `%MktEx` something of a two-stage functionality. At first, it worries about eliminating twos and also having the right number, but mostly it eliminates twos. Once it eliminates the most-penalized source of badness then it can concentrate on the remaining source, as long as it does not bring back any of the badness it previously eliminated. Often times, it does not matter which component you weight the most, nor does it matter how much you weight them, as long as each component has a differential weight.

We could run the `%ChoicEff` macro again to see what the final *D*-efficiency is like before as follows:

```
%choiceff(data=nodups,             /* candidate set of choice sets         */
          model=class(x1-x20 / zero=first), /* main effects, first ref level*/
          seed=495,                /* random number seed                   */
          maxiter=10,              /* maximum iterations for each phase    */
          nsets=18,                /* number of choice sets                */
          nalts=6,                 /* number of alternatives               */
          options=nodups,          /* no duplicate choice sets             */
          beta=zero)               /* assumed beta vector, Ho: b=0         */
```

The summary table is as follows:

```
                                Final Results

                          Design                 3
                          Choice Sets           18
                          Alternatives           6
                          Parameters            20
                          Maximum Parameters    90
                          D-Efficiency      1.0531
                          D-Error           0.9495
```

These results are similar to those from before. We could also specify the standardized orthogonal contrast coding as follows:

```
%choiceff(data=nodups,             /* candidate set of choice sets         */
          model=class(x1-x20 / sta),/* model with stdzd orthogonal coding   */
          seed=495,                /* random number seed                   */
          maxiter=10,              /* maximum iterations for each phase    */
          nsets=18,                /* number of choice sets                */
          nalts=6,                 /* number of alternatives               */
          options=nodups           /* no duplicate choice sets             */
                  relative,        /* display relative D-efficiency        */
          beta=zero)               /* assumed beta vector, Ho: b=0         */
```

The summary table is as follows:

```
                        Final Results

                  Design                  3
                  Choice Sets            18
                  Alternatives            6
                  Parameters             20
                  Maximum Parameters     90
                  D-Efficiency       4.2126
                  Relative D-Eff    23.4032
                  D-Error            0.2374
                  1 / Choice Sets    0.0556
```

D-efficiency is 23.4032. The largest it could possibly be is 25 since 5 of 20 attributes (25%) vary.

Five-Level Factors; Partial Profiles Constructed Using Restrictions

This next example extends what we discussed in the previous example and constructs a somewhat different style of partial-profile design. This design has five alternatives and 15 five-level factors, five of which vary in each choice set. Unlike the previous example, however, the constant factors are not all at the base-line level. The constant factors can have any of the levels, and we use the first factor within each attribute when we check the restrictions. The partial-profile restrictions macro along with `%MktEx` code, which uses the same basic option set that we used in the previous example, is as follows:

```
%macro partprof;
   sum = 0;
   do k = 1 to 15;
      sum = sum + (x[k+15] = x[k] & x[k+30] = x[k] &
                   x[k+45] = x[k] & x[k+60] = x[k]);
      end;
   bad = abs(sum - 10);
   if sum < 10 & x[j1] ^= x[mod(j1 - 1, 15) + 1] then bad = bad + 1000;
   %mend;

%mktex(5 ** 75,                   /* 75 five-level factors               */
       n=400,                     /* 400 runs                            */
       order=random,              /* loop over columns in a random order */
       out=cand,                  /* output design                       */
       restrictions=partprof,     /* name of restrictions macro          */
       seed=472,                  /* random number seed                  */
       maxtime=0,                 /* stop once design conforms           */
       options=largedesign        /* make large design more quickly      */
               nosort             /* do not sort output design           */
               resrep             /* detailed report on restrictions     */
               quickr             /* very quick run with random init     */
               nox)               /* suppress x1, x2, ... creation       */
```

The macro counts the number of times all of the linear factors in a choice set attribute are constant within choice set, that is they equal the level of the first linear factor in the attribute, then it computes badness in the customary way. Also like before, when not all restrictions are met, any level that is not equal to the first factor within its attribute is heavily penalized. The macro uses the "whack" approach to impose constant attributes. Some of the iteration history is as follows:

```
                          Algorithm Search History

                                Current          Best
Design    Row,Col  D-Efficiency  D-Efficiency  Notes
----------------------------------------------------------
     1       1          58.6862                0 Violations
     1       2          58.7150                1 Violations
     1       3          58.6810                2 Violations
     .
     .
     .
     1     100          56.0416                5 Violations
     1     101          55.9496                0 Violations
     1     102          55.8855                4 Violations
     .
     .
     .
     1     200          48.0539                3 Violations
     1     201          47.9776                3 Violations
     1     202          47.9279                4 Violations
     .
     .
     .
     1     300          38.9711                0 Violations
     1     301          38.8695                4 Violations
     1     302          38.7805                3 Violations
     .
     .
     .
     1     398          30.4723                2 Violations
     1     399          30.4075                2 Violations
     1     400          30.3169                2 Violations
     1       1          30.4110                0 Violations
     1       2          30.4491                0 Violations
     1       3          30.5001                0 Violations
     .
     .
     .
```

```
   1     73           32.8050                 2 Violations
   1     73           32.8708                 0 Violations
   1     74           32.8840                 0 Violations
   1     75           32.8518                 1 Violations
   1     75           32.8845                 0 Violations
   .
   .
   .

   1    398           34.9958                 0 Violations
   1    399           35.0104                 0 Violations
   1    400           34.9901                 0 Violations
   1      1           35.0015                 0 Violations
   1      2           35.0133                 0 Violations
   1      3           35.0337                 0 Violations
   .
   .
   .

   1    398           43.0865                 0 Violations
   1    399           43.1054                 0 Violations
   1    400           43.1304                 0 Violations
   1      1   1       43.1304       43.1304  Conforms
   1      1  42       43.1311       43.1311
   1      1  17       43.1338       43.1338
   1        End       43.1305
```

This iteration history has a pattern very similar to what we saw in the previous example with the nudge approach. In the first pass through the design, not all restrictions are met. Even the whack approach does not guarantee that all restrictions get met right away. In the second pass, some rows are processed more than once until all restrictions are met. We can see that in choice sets 73 and 75. By the end of the second pass, all restrictions are met (efficiency starts back up), `%MktEx` realizes all restrictions are met at the end of the third pass, and then it stops. The `%MktEx` macro iterates longer if we do not specify `options=largedesign`, but for now when you are testing a new restrictions macro, it is good to check the results before `%MktEx` spends a long time iterating. The following steps create a choice design from this linear candidate set in the familiar way:

```
%mktkey(5 15)

%mktroll(design=cand, key=key, out=rolled)

%mktdups(generic, data=rolled, out=nodups, factors=x1-x15, nalts=5)

%choiceff(data=nodups,              /* candidate set of choice sets         */
         model=class(x1-x15 / sta),/* model with stdz orthogonal coding    */
         seed=513,                 /* random number seed                   */
         maxiter=10,               /* maximum iterations for each phase    */
         nsets=15,                 /* number of choice sets                */
         nalts=5,                  /* number of alternatives               */
         options=nodups            /* no duplicate choice sets             */
                 relative,         /* display relative D-efficiency        */
         beta=zero)                /* assumed beta vector, Ho: b=0         */

proc print data=best(obs=15); id set; by notsorted set; var x1-x15; run;
```

The key with 5 rows, 15 columns and the variable names `x1 - x75` is as follows:

```
      x1   x2   x3   x4   x5   x6   x7   x8   x9   x10  x11  x12  x13  x14  x15

      x1   x2   x3   x4   x5   x6   x7   x8   x9   x10  x11  x12  x13  x14  x15
      x16  x17  x18  x19  x20  x21  x22  x23  x24  x25  x26  x27  x28  x29  x30
      x31  x32  x33  x34  x35  x36  x37  x38  x39  x40  x41  x42  x43  x44  x45
      x46  x47  x48  x49  x50  x51  x52  x53  x54  x55  x56  x57  x58  x59  x60
      x61  x62  x63  x64  x65  x66  x67  x68  x69  x70  x71  x72  x73  x74  x75
```

Skipping the `%ChoicEff` macro output for a moment, part of the choice design is as follows:

```
        Set  x1  x2  x3  x4  x5  x6  x7  x8  x9  x10  x11  x12  x13  x14  x15

         33   5   5   4   5   2   1   4   2   3   5    5    2    5    3    1
              5   1   3   1   2   1   4   4   3   5    1    2    5    3    1
              5   5   3   4   2   1   4   5   3   5    2    2    5    3    1
              5   1   1   2   2   1   4   1   3   5    5    2    5    3    1
              5   2   5   3   2   1   4   3   3   5    3    2    5    3    1

        291   3   2   3   1   2   4   5   4   2   4    3    1    1    2    4
              3   2   3   1   2   4   5   3   3   4    3    1    5    4    4
              3   2   3   1   2   4   4   2   5   4    3    1    3    5    4
              3   2   3   1   2   4   5   5   2   4    3    1    4    3    4
              3   2   3   1   2   4   4   1   4   4    3    1    1    1    4
```

```
186   3   4   3   3   5   2   1   1   4   1    3    1    1    3    5
      3   4   5   3   5   2   5   2   4   2    3    1    1    3    4
      3   4   2   3   5   2   3   1   4   4    3    1    1    3    2
      3   4   1   3   5   2   4   3   4   5    3    1    1    3    5
      3   4   4   3   5   2   4   5   4   3    3    1    1    3    3
```

The pattern of constant and nonconstant attributes looks correct: 10 constant and 5 nonconstant attributes per choice set. Furthermore, the constant attributes have the full range of levels. The last part of the output from the `%ChoicEff` macro is as follows:

```
                              Final Results

                        Design                   9
                        Choice Sets             15
                        Alternatives             5
                        Parameters              60
                        Maximum Parameters      60
                        D-Efficiency        2.3286
                        Relative D-Eff     15.5243
                        D-Error             0.4294
                        1 / Choice Sets     0.0667

            Variable                                      Standard
      n       Name       Label     Variance     DF          Error

      1       x11        x1 1       1.24427      1        1.11547
      2       x12        x1 2       1.37689      1        1.17341
      3       x13        x1 3       1.62664      1        1.27540
      4       x14        x1 4       1.25242      1        1.11912
      5       x21        x2 1       1.06398      1        1.03149
      6       x22        x2 2       2.50082      1        1.58140
      7       x23        x2 3       1.54702      1        1.24379
      8       x24        x2 4       1.00564      1        1.00282
      9       x31        x3 1       1.68441      1        1.29785
     10       x32        x3 2       0.72847      1        0.85351
     11       x33        x3 3       1.58669      1        1.25964
     12       x34        x3 4       1.36204      1        1.16707
     13       x41        x4 1       1.57492      1        1.25496
     14       x42        x4 2       2.40504      1        1.55082
     15       x43        x4 3       2.07707      1        1.44120
     16       x44        x4 4       1.37807      1        1.17391
     17       x51        x5 1       1.84721      1        1.35912
     18       x52        x5 2       3.63100      1        1.90552
     19       x53        x5 3       2.87391      1        1.69526
     20       x54        x5 4       0.87844      1        0.93725
```

```
21      x61      x6 1      2.42661     1      1.55776
22      x62      x6 2      3.33545     1      1.82632
23      x63      x6 3      2.00194     1      1.41490
24      x64      x6 4      2.13206     1      1.46016
25      x71      x7 1      0.82550     1      0.90857
26      x72      x7 2      1.15680     1      1.07555
27      x73      x7 3      0.80379     1      0.89655
28      x74      x7 4      0.76404     1      0.87409
29      x81      x8 1      2.98038     1      1.72638
30      x82      x8 2      1.37852     1      1.17410
31      x83      x8 3      2.82780     1      1.68161
32      x84      x8 4      2.81404     1      1.67751
33      x91      x9 1      1.00980     1      1.00489
34      x92      x9 2      0.98026     1      0.99008
35      x93      x9 3      1.30321     1      1.14158
36      x94      x9 4      2.07571     1      1.44073
37      x101     x10 1     1.66894     1      1.29187
38      x102     x10 2     0.69715     1      0.83496
39      x103     x10 3     0.83388     1      0.91317
40      x104     x10 4     1.44343     1      1.20143
41      x111     x11 1     0.77932     1      0.88279
42      x112     x11 2     2.22471     1      1.49155
43      x113     x11 3     0.45386     1      0.67369
44      x114     x11 4     1.50188     1      1.22551
45      x121     x12 1     1.69913     1      1.30351
46      x122     x12 2     1.18246     1      1.08741
47      x123     x12 3     0.83460     1      0.91357
48      x124     x12 4     0.99096     1      0.99547
49      x131     x13 1     4.57118     1      2.13803
50      x132     x13 2     1.51438     1      1.23060
51      x133     x13 3     3.21429     1      1.79284
52      x134     x13 4     8.78785     1      2.96443
53      x141     x14 1     1.22443     1      1.10654
54      x142     x14 2     1.85878     1      1.36337
55      x143     x14 3     1.39724     1      1.18205
56      x144     x14 4     0.91801     1      0.95813
57      x151     x15 1     1.27954     1      1.13117
58      x152     x15 2     3.05334     1      1.74738
59      x153     x15 3     3.27077     1      1.80853
60      x154     x15 4     2.76477     1      1.66276
                                            ==
                                            60
```

The relative D-efficiency is 15.5243. The largest it could possibly be is 33 since 5 of 15 attributes (33%) vary. Our value is on the order of half that. Another way to assess the quality of the design is to look at the parameter variances. In this table they seem quite variable. This is usually not a good sign. This run of the `%ChoicEff` macro requested a design with only 15 choice sets, which is not a lot. In fact, with 15 choice sets and 4 alternatives, we can estimate at most 60 parameters, which is exactly the number we are trying to estimate here. This is usually not a good idea. Let's try again, this time with 30 choice sets. The following step finds the design:

```
%choiceff(data=nodups,              /* candidate set of choice sets         */
          model=class(x1-x15 / sta),/* model with stdz orthogonal coding    */
          seed=513,                 /* random number seed                   */
          maxiter=10,               /* maximum iterations for each phase    */
          nsets=30,                 /* number of choice sets                */
          nalts=5,                  /* number of alternatives               */
          options=nodups            /* no duplicate choice sets             */
                  relative,         /* display relative D-efficiency        */
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

The new results and variance tables are as follows:

```
                              Partial Profiles

                               Final Results

                         Design                 1
                         Choice Sets           30
                         Alternatives           5
                         Parameters            60
                         Maximum Parameters   120
                         D-Efficiency      7.7251
                         Relative D-Eff   25.7503
                         D-Error           0.1294
                         1 / Choice Sets   0.0333

                              Partial Profiles

                 Variable                                    Standard
            n      Name       Label     Variance    DF         Error

            1      x11        x1 1       0.16200     1       0.40250
            2      x12        x1 2       0.15231     1       0.39027
            3      x13        x1 3       0.15801     1       0.39751
            4      x14        x1 4       0.18802     1       0.43361
            5      x21        x2 1       0.20554     1       0.45337
            6      x22        x2 2       0.15363     1       0.39196
            7      x23        x2 3       0.16931     1       0.41147
            8      x24        x2 4       0.17124     1       0.41381
            9      x31        x3 1       0.14382     1       0.37924
           10      x32        x3 2       0.13760     1       0.37094
           11      x33        x3 3       0.19570     1       0.44238
           12      x34        x3 4       0.15172     1       0.38952
           13      x41        x4 1       0.13177     1       0.36300
           14      x42        x4 2       0.13093     1       0.36185
           15      x43        x4 3       0.13788     1       0.37133
```

```
16      x44      x4 4     0.15115     1     0.38878
17      x51      x5 1     0.14697     1     0.38336
18      x52      x5 2     0.12922     1     0.35947
19      x53      x5 3     0.12505     1     0.35362
20      x54      x5 4     0.17476     1     0.41804
21      x61      x6 1     0.15004     1     0.38735
22      x62      x6 2     0.17357     1     0.41661
23      x63      x6 3     0.12204     1     0.34934
24      x64      x6 4     0.13766     1     0.37102
25      x71      x7 1     0.18417     1     0.42915
26      x72      x7 2     0.16833     1     0.41027
27      x73      x7 3     0.13836     1     0.37197
28      x74      x7 4     0.14091     1     0.37538
29      x81      x8 1     0.16784     1     0.40969
30      x82      x8 2     0.16454     1     0.40564
31      x83      x8 3     0.14048     1     0.37481
32      x84      x8 4     0.17935     1     0.42349
33      x91      x9 1     0.17887     1     0.42293
34      x92      x9 2     0.17178     1     0.41447
35      x93      x9 3     0.18717     1     0.43263
36      x94      x9 4     0.16287     1     0.40357
37      x101     x10 1    0.17880     1     0.42284
38      x102     x10 2    0.15523     1     0.39400
39      x103     x10 3    0.15494     1     0.39362
40      x104     x10 4    0.15128     1     0.38894
41      x111     x11 1    0.15144     1     0.38916
42      x112     x11 2    0.19196     1     0.43813
43      x113     x11 3    0.18085     1     0.42526
44      x114     x11 4    0.17372     1     0.41680
45      x121     x12 1    0.16954     1     0.41175
46      x122     x12 2    0.19912     1     0.44623
47      x123     x12 3    0.16908     1     0.41120
48      x124     x12 4    0.15868     1     0.39834
49      x131     x13 1    0.17289     1     0.41580
50      x132     x13 2    0.19481     1     0.44138
51      x133     x13 3    0.21682     1     0.46564
52      x134     x13 4    0.17501     1     0.41834
53      x141     x14 1    0.15621     1     0.39524
54      x142     x14 2    0.13804     1     0.37154
55      x143     x14 3    0.13537     1     0.36793
56      x144     x14 4    0.16000     1     0.40000
57      x151     x15 1    0.14634     1     0.38254
58      x152     x15 2    0.16575     1     0.40712
59      x153     x15 3    0.14842     1     0.38525
60      x154     x15 4    0.17341     1     0.41643
                                       ==
                                       60
```

This looks much better. The variances are smaller and more uniform and relative *D*-efficiency is larger. Note that it is good to run the `%MktEx` macro again without `options=largedesign` and let it iterate more before making the final choice design.

Next, we will investigate another thing you can try. Typically, the `%MktEx` macro is run so that it loops over all of the columns in a row, and then it goes on to the next row. Alternatively, it can work with *pairs* of columns at one time using the `exchange=2` option. Working with pairs of columns instead of single columns is always much slower, but sometimes it can make better designs. The following steps create the design:

```
%macro partprof;
   sum = 0;
   do k = 1 to 15;
      sum = sum + (x[k+15] = x[k] & x[k+30] = x[k] &
                   x[k+45] = x[k] & x[k+60] = x[k]);
      end;
   bad = abs(sum - 10);
   if sum < 10 then do;
      if x[j1] ^= x[mod(j1 - 1, 15) + 1] then bad = bad + 1000;
      if x[j2] ^= x[mod(j2 - 1, 15) + 1] then bad = bad + 1000;
      end;
   %mend;

%mktex(5 ** 75,                     /* 75 five-level factors              */
       n=400,                       /* 400 runs                           */
       optiter=0,                   /* no PROC OPTEX iterations           */
       tabiter=0,                   /* no OA initialization iterations    */
       maxtime=720,                 /* at most 720 minutes per phase      */
       order=random=15,             /* pairwise exchanges within          */
                                    /*    nonconstant attributes          */
       out=sasuser.cand,            /* output design stored permanently   */
       restrictions=partprof,       /* name of restrictions macro         */
       seed=472,                    /* random number seed                 */
       exchange=2,                  /* pairwise exchanges                 */
       maxstages=1,                 /* maximum number of algorithm stages */
       options=largedesign          /* make large design more quickly     */
               nosort               /* do not sort output design          */
               resrep               /* detailed report on restrictions    */
               nox)                 /* suppress x1, x2, ... creation      */
```

The initial quantification of badness is the same. Like before, nonconforming levels are whacked. This time however, they are whacked in two ways—when `j1`, the primary column index points to a nonconstant level, and when `j2`, the secondary column index for the pairwise exchange indexes a nonconstant level. In the `%MktEx` invocation, we now see `maxtime=720` so that `%MktEx` can run over night for 12 hours (or 720 minutes). We also see `exchange=2` for pairwise exchanges. The output data set is stored as a permanent SAS data set in the `sasuser` library. If we search for 12 hours for a design, we want to make sure it is there for us if we accidentally trip over the power cord in the morning before we have our coffee. See page 309 for more information about permanent SAS data sets.

There is one more option in this example that we have not used previously, `order=random=15`. This is a special variation on `order=random` for pairwise exchanges in partial-profile designs. Before this option is explained, here is a bit of background. Sequential pairwise exchanges for m factors works like this: `%MktEx` sets `j1` = 1, 2, 3, ..., m and `j2` = `j1` + 1, `j1` + 2, ..., m. Together, the variables `j1` and `j2` loop over all pairs of columns. Random pairwise exchanges work like this: this: `%MktEx` sets `j1` = random_permutation(1, 2, 3, ..., m) and `j2` = random_permutation(`j1` + 1, `j1` + 2, ..., m). Together, the variables `j1` and `j2` loop over all pairs of columns but in a random order. For partial profiles, pairwise exchanges are appealing, because sometimes there is a lot to be gained by having two values change at once. However, it does not make sense to consider simultaneously changing the level of a nonconstant attribute and the level of a constant attribute, nor does it make sense to consider pairwise exchanges within constant attributes. Random exchange with a value of 15 specified works like this: `%MktEx` sets `j1` = random_permutation(1, 2, 3, ..., m) and `j2` is set to a sequential list of the other factors in the same attribute as `j1`. For example, when `j1` = 18, which means `j1` is indexing the second alternative (18 is in the second block of 15 factors) for the third attribute (18 is 3 beyond the fifteenth factor, which is the end of the first block), then `j2` = 3, 18, 33, 48, and 63 for a nonconstant third attribute (which index the 5 factors that make up the third attribute) and `j2` = `j1` for constant attributes. This does pairwise exchanges but only within nonconstant attributes. This eliminates a lot of unproductive pairs from consideration.

A small part of the iteration history is as follows:

```
                      Algorithm Search History

                                Current          Best
Design    Row,Col  D-Efficiency  D-Efficiency  Notes
----------------------------------------------------------
     1      Start       58.6611                Ran,Mut,Ann
     1      1           58.6962                0 Violations
     1      2           58.7140                0 Violations
     1      3           58.7171                1 Violations
     1      4           58.7076                1 Violations
     1      5           58.7269                2 Violations
     1      6           58.6943                2 Violations
     .
     .
     .

     1    398           31.1040                0 Violations
     1    399           31.0346                3 Violations
     1    400           30.9260                1 Violations
     1      1           31.0114                0 Violations
     1      2           31.0566                0 Violations
     1      3           31.1412                0 Violations
     .
     .
     .
```

```
1     48           32.6265                0 Violations
1     49           32.6150                3 Violations
1     49           32.5890                1 Violations
1     49           32.6547                0 Violations
1     50           32.7205                0 Violations
.
.
.
1    398           34.7853                0 Violations
1    399           34.7463                0 Violations
1    400           34.7763                0 Violations
1      1           34.8061                0 Violations
1      2           34.8381                0 Violations
1      3           34.8520                0 Violations
.
.
.
1    398           43.2881                0 Violations
1    399           43.2774                0 Violations
1    400           43.3051                0 Violations
1      1   1       43.3051       43.3051  Conforms
1      1  72       43.3085       43.3085
1      1  26       43.3105       43.3105
1      1  41       43.3113       43.3113
.
.
.
1    154   7       50.0432       50.0432
1    176   1       50.0432       50.0432
1    217  17       50.0432       50.0432
1        End       50.0432
```

It is followed by the following messages:

```
NOTE: Stopping early, possibly before convergence, with a large design.
NOTE: Quitting the algorithm search step after 720.01 minutes and 22 designs.
```

The following steps make and evaluate the choice design:

```
%mktkey(5 15)

%mktroll(design=sasuser.cand, key=key, out=rolled)

%mktdups(generic, data=rolled, out=nodups, factors=x1-x15, nalts=5)

%choiceff(data=nodups,              /* candidate set of choice sets         */
          model=class(x1-x15 / sta),/* model with stdz orthogonal coding    */
          seed=513,                 /* random number seed                   */
          maxiter=10,               /* maximum iterations for each phase    */
          nsets=30,                 /* number of choice sets                */
          nalts=5,                  /* number of alternatives               */
          options=nodups            /* no duplicate choice sets             */
                  relative,         /* display relative D-efficiency        */
          beta=zero)                /* assumed beta vector, Ho: b=0         */

proc print data=best(obs=15); id set; by notsorted set; var x1-x15; run;
```

The new results and variance tables are as follows:

```
                              Partial Profiles

                               Final Results

                          Design                 3
                          Choice Sets           30
                          Alternatives           5
                          Parameters            60
                          Maximum Parameters   120
                          D-Efficiency      8.1495
                          Relative D-Eff   27.1650
                          D-Error           0.1227
                          1 / Choice Sets   0.0333

                              Partial Profiles

                 Variable                                   Standard
            n      Name      Label    Variance    DF        Error

            1      x11       x1 1      0.15201     1      0.38989
            2      x12       x1 2      0.11803     1      0.34356
            3      x13       x1 3      0.13012     1      0.36072
            4      x14       x1 4      0.11936     1      0.34548
            5      x21       x2 1      0.15472     1      0.39334
            6      x22       x2 2      0.14263     1      0.37766
            7      x23       x2 3      0.17875     1      0.42279
            8      x24       x2 4      0.17528     1      0.41866
            9      x31       x3 1      0.13811     1      0.37163
           10      x32       x3 2      0.15821     1      0.39775
```

```
11      x33     x3 3    0.13531     1     0.36784
12      x34     x3 4    0.13226     1     0.36367
13      x41     x4 1    0.14763     1     0.38422
14      x42     x4 2    0.14864     1     0.38554
15      x43     x4 3    0.13789     1     0.37134
16      x44     x4 4    0.14818     1     0.38494
17      x51     x5 1    0.15596     1     0.39491
18      x52     x5 2    0.15623     1     0.39526
19      x53     x5 3    0.15559     1     0.39444
20      x54     x5 4    0.13505     1     0.36750
21      x61     x6 1    0.14271     1     0.37777
22      x62     x6 2    0.14806     1     0.38479
23      x63     x6 3    0.14098     1     0.37548
24      x64     x6 4    0.14159     1     0.37629
25      x71     x7 1    0.14727     1     0.38376
26      x72     x7 2    0.11428     1     0.33806
27      x73     x7 3    0.13768     1     0.37105
28      x74     x7 4    0.14633     1     0.38253
29      x81     x8 1    0.12268     1     0.35026
30      x82     x8 2    0.13987     1     0.37399
31      x83     x8 3    0.13153     1     0.36267
32      x84     x8 4    0.12463     1     0.35303
33      x91     x9 1    0.21901     1     0.46798
34      x92     x9 2    0.16280     1     0.40349
35      x93     x9 3    0.14938     1     0.38650
36      x94     x9 4    0.14706     1     0.38348
37      x101    x10 1   0.15695     1     0.39617
38      x102    x10 2   0.17411     1     0.41726
39      x103    x10 3   0.14308     1     0.37826
40      x104    x10 4   0.13558     1     0.36822
41      x111    x11 1   0.14586     1     0.38192
42      x112    x11 2   0.15817     1     0.39771
43      x113    x11 3   0.15272     1     0.39080
44      x114    x11 4   0.19799     1     0.44496
45      x121    x12 1   0.14785     1     0.38451
46      x122    x12 2   0.21303     1     0.46155
47      x123    x12 3   0.14543     1     0.38136
48      x124    x12 4   0.12144     1     0.34848
49      x131    x13 1   0.19242     1     0.43866
50      x132    x13 2   0.20030     1     0.44755
51      x133    x13 3   0.17448     1     0.41770
52      x134    x13 4   0.17275     1     0.41563
53      x141    x14 1   0.15431     1     0.39282
54      x142    x14 2   0.13919     1     0.37309
55      x143    x14 3   0.15637     1     0.39543
```

```
        56       x144       x14 4      0.13149       1       0.36261
        57       x151       x15 1      0.15171       1       0.38950
        58       x152       x15 2      0.14736       1       0.38387
        59       x153       x15 3      0.14358       1       0.37892
        60       x154       x15 4      0.15097       1       0.38855
                                                    ==
                                                    60
```

Now, relative *D*-Efficiency is 27.1650 compared to 25.7503 previously. This is typical in the sense that it is not unusual to get a small gain for a large increase in computation. The first few choice sets are as follows:

```
Set  x1  x2  x3  x4  x5  x6  x7  x8  x9  x10  x11  x12  x13  x14  x15

128   1   3   5   3   4   1   5   1   5   2    3    3    3    2    4
      4   3   3   5   4   1   2   1   5   3    3    3    3    2    4
      5   3   1   1   4   1   3   1   5   4    3    3    3    2    4
      3   3   2   4   4   1   4   1   5   1    3    3    3    2    4
      2   3   4   2   4   1   1   1   5   5    3    3    3    2    4

100   1   4   3   4   3   4   5   3   2   2    4    4    4    1    5
      5   4   5   4   4   5   5   1   2   2    4    4    4    1    5
      4   4   2   4   5   2   5   2   2   2    4    4    4    1    5
      2   4   1   4   1   3   5   4   2   2    4    4    4    1    5
      3   4   4   4   2   1   5   5   2   2    4    4    4    1    5

 11   3   1   3   4   1   1   4   4   2   4    1    3    4    1    5
      3   1   4   4   1   1   2   2   5   4    4    3    4    1    5
      3   1   5   4   1   1   3   5   1   4    5    3    4    1    5
      3   1   2   4   1   1   1   1   4   4    2    3    4    1    5
      3   1   1   4   1   1   5   3   3   4    3    3    4    1    5
```

The example starting on page 613 and the choice design shown on page 618 creates a partial-profile design with all constant attributes equal to one. In contrast, this design uses the full range of values for the constant attributes. In terms of fitting the choice model, it does not matter. An attribute that is constant within a choice set does not contribute to the likelihood function for that choice set, and this is true no matter what the constant value is. It typically does not matter for data collection either, since data collection is typically phrased in terms like "everything else being equal" without any specifics about what the equal levels are. About the only difference is in the `%MktEx` macro. *D*-efficiency should be higher with the varying-constant approach than with the all-one approach.

Partial Profiles from Block Designs and Orthogonal Arrays

These next examples make optimal partial-profile designs from balanced incomplete block designs (BIBDs) and small orthogonal arrays (OAs). Before we get into partial profiles, let's discuss BIBDs and review OAs.

Balanced Incomplete Block Designs

The following design is a BIBD:

```
                    Balanced Incomplete Block Design

                           x1    x2    x3    x4

                           10     4    16     1
                           16    13    14    11
                           12    15     4    14
                            8    10     2    13
                           13     9     1    15
                            4     6    13     5
                            7     5    15    10
                            5    12     8    16
                            9    14     5     2
                            3     5    11     1
                            7    13     3    12
                            2    15     3    16
                           14     6    10     3
                            8     3     9     4
                           10    11     9    12
                            1     7    14     8
                            2     1    12     6
                           11     2     4     7
                            6    16     7     9
                           15     8     6    11
```

It consists of $b = 20$ rows or *blocks*. The design has $k = 4$ columns, so each block is of size four. The design consists of the integers 1 to 16, so over the entire design, there are $t = 16$ attributes. The design has a total of $bk = 20 \times 4 = 80$ elements. This design is balanced in the sense that each of the $t = 16$ attributes occurs exactly $r = 5$ times ($rt = 5 \times 16 = bk = 20 \times 4 = 80$). Furthermore, the $t(t-1)/2 = 16(16-1)/2 = bk(k-1)/2 = 20 \times 4(4-1)/2 = 120$ pairs of attributes occur together in the same block exactly once. This is what makes this design a BIBD. The attribute by attribute frequencies are as follows:

```
                  Attribute by Attribute Frequencies

       1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16

  1    5  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  2       5  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  3          5  1  1  1  1  1  1  1  1  1  1  1  1  1
  4             5  1  1  1  1  1  1  1  1  1  1  1  1
  5                5  1  1  1  1  1  1  1  1  1  1  1
  6                   5  1  1  1  1  1  1  1  1  1  1
  7                      5  1  1  1  1  1  1  1  1  1
  8                         5  1  1  1  1  1  1  1  1
  9                            5  1  1  1  1  1  1  1
 10                               5  1  1  1  1  1  1
 11                                  5  1  1  1  1  1
 12                                     5  1  1  1  1
 13                                        5  1  1  1
 14                                           5  1  1
 15                                              5  1
 16                                                 5
```

In the partial-profile context, this BIBD instructs us to create 20 blocks of choice sets. This BIBD can be used for the situation where there are 16 attributes and 4 are varied at a time. In the first block, attributes 10, 4, 16, and 1 vary while the remaining attributes are held constant; in the second block, attributes 16, 13, 14, and 11 vary while the remaining attributes are held constant; and so on. Every attribute appears in a block of choice sets with every other attribute and each attribute occurs exactly as often as every other attribute. A BIBD is not an orthogonal array or ordinary factorial design like we get out of the `%MktEx` macro, so we use a different tool, the `%MktBIBD` macro to find BIBDs. For more information about BIBDs and how they relate to factorial designs see page 965. BIBDs only exist for specific and limited combinations of b, k, and t. We can run the `%MktBSize` macro to find sizes that meet necessary but not sufficient criteria for the existence of a BIBD as follows:

```
%mktbsize(b=20, nattrs=16, setsize=4)
```

This step reports the following, which shows that a BIBD might exist for this specification, but of course we already knew that:

```
                                                               n
t             k      b          r            Lambda          Total
Number of     Set    Number     Attribute    Pairwise        Sample
Attributes    Size   of Sets    Frequency    Frequencies     Size

    16          4       20          5              1            80
```

More generally, we can run the `%MktBSize` macro with lists of parameters and get a list of those combinations of parameters that meet the necessary but not sufficient conditions for the existence of a BIBD. This list includes the design we just saw. The following step generates the list:

```
   %mktbsize(nattrs=15 to 20, setsize=3 to t / 2, b=t to 30)
```

The results are as follows:

```
                                                                    n
         t              k        b          r            Lambda         Total
         Number of      Set      Number     Attribute    Pairwise       Sample
         Attributes     Size     of Sets    Frequency    Frequencies    Size

             15            5        21            7            2           105
             15            7        15            7            3           105
             16            4        20            5            1            80
             16            6        16            6            2            96
             16            8        30           15            7           240
             19            9        19            9            4           171
```

There is no guarantee that `%MktBIBD` will find a BIBD for any specification, even when one is in fact known to exist. However, it usually does quite well in finding smaller BIBDS and in finding designs that are close the rest of the time. When it cannot find a BIBD, it finds a design where each attribute occurs as often as every other attribute or almost as often and each pair of attributes occurs together almost equally often. Furthermore, the `%MktBIBD` macro can find designs like this even when the parameters do not meet the necessary criteria. For the purposes of making a partial-profile design, a block design with almost equal frequencies is usually good enough. The following step shows how we found the BIBD shown in the beginning of this section:

```
   %mktbibd(b=20, nattrs=16, setsize=4, seed=17)
```

The entire set of output from this macro is as follows:

```
                    Block Design Efficiency Criterion       100.0000
                    Number of Attributes, t                       16
                    Set Size, k                                    4
                    Number of Sets, b                             20
                    Attribute Frequency                            5
                    Pairwise Frequency                             1
                    Total Sample Size                             80
                    Positional Frequencies Optimized?            Yes
```

```
                     Attribute by Attribute Frequencies

       1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16

   1   5   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
   2       5   1   1   1   1   1   1   1   1   1   1   1   1   1   1
   3           5   1   1   1   1   1   1   1   1   1   1   1   1   1
   4               5   1   1   1   1   1   1   1   1   1   1   1   1
   5                   5   1   1   1   1   1   1   1   1   1   1   1
   6                       5   1   1   1   1   1   1   1   1   1   1
   7                           5   1   1   1   1   1   1   1   1   1
   8                               5   1   1   1   1   1   1   1   1
   9                                   5   1   1   1   1   1   1   1
  10                                       5   1   1   1   1   1   1
  11                                           5   1   1   1   1   1
  12                                               5   1   1   1   1
  13                                                   5   1   1   1
  14                                                       5   1   1
  15                                                           5   1
  16                                                               5

                      Attribute by Position Frequencies

                               1   2   3   4

                           1   1   1   1   2
                           2   2   1   1   1
                           3   1   1   2   1
                           4   1   1   2   1
                           5   1   2   1   1
                           6   1   2   1   1
                           7   2   1   1   1
                           8   2   1   1   1
                           9   1   1   2   1
                          10   2   1   1   1
                          11   1   1   1   2
                          12   1   1   1   2
                          13   1   2   1   1
                          14   1   1   2   1
                          15   1   2   1   1
                          16   1   1   1   2
```

```
                    Balanced Incomplete Block Design

                           x1    x2    x3    x4

                           10     4    16     1
                           16    13    14    11
                           12    15     4    14
                            8    10     2    13
                           13     9     1    15
                            4     6    13     5
                            7     5    15    10
                            5    12     8    16
                            9    14     5     2
                            3     5    11     1
                            7    13     3    12
                            2    15     3    16
                           14     6    10     3
                            8     3     9     4
                           10    11     9    12
                            1     7    14     8
                            2     1    12     6
                           11     2     4     7
                            6    16     7     9
                           15     8     6    11
```

The first part of the output consists of a "factoid" of miscellaneous statistics, parameters, and information. The efficiency criterion is 100, so a BIBD was found. Next, the parameters of the BIBD, $t = 16$, $k = 4$, and $b = 20$ are reported. The attribute frequency of 5 shows that each of the $t = 16$ attributes occurs five times in the array. Each pair of attributes occurs once. The total sample size refers to the $bk = 20 \times 4 = 80$ elements in the BIBD. In the final BIBD, the positional frequencies are optimized. This is shown in the attribute by position frequencies, which are discussed soon.

The factoid is followed by the attribute by attribute frequency matrix. We want to see a constant diagonal and a constant upper triangular matrix above the diagonal, and that is what we get since we have a BIBD. The next matrix contains the attribute by position frequencies. We see that each attribute occurs in each position either one or two times. It is impossible for the results to be constant in this BIBD. However, they are as close to constant as they possibly can be, and there are no zeros, threes, or greater values. Note though, that the attribute by position frequencies are not important to us in the partial-profile context. We could specify `positer=` to turn off the iterations that optimize the positional frequencies to make the macro run faster for this problem. In this case, we get the same BIBD, but with the values in a different order within row. The last matrix is the BIBD.

Orthogonal Arrays

An orthogonal array (OA) is a factorial design in which all estimable effects are uncorrelated. The `%MktEx` macro has an extensive catalog of orthogonal arrays. A sample of a small OA is as follows:

```
                                 x1    x2    x3    x4    x5

                                  1     1     1     1     1
                                  1     2     2     2     2
                                  2     1     2     1     2
                                  2     2     1     2     1
                                  3     1     1     2     2
                                  3     2     2     1     1
                                  4     1     2     2     1
                                  4     2     1     1     2
```

This design is balanced—each of the four levels of `x1` occurs exactly twice, and each of the two levels of `x2-x4` occurs exactly four times. This design is orthogonal—each of the eight pairs of levels of `x1` with `x2-x5` occurs exactly once, and each of the four pairs of levels for each of the pairs of factors in `x2-x5` occurs exactly twice. Hence this design is an OA. It can be created as follows:

```
%mktex(4 2**4, n=8)

proc print noobs; run;
```

80 Choice Sets, 16 Binary Attributes, Four Varying

In this example, we create a partial-profile design with $t = 16$ binary attributes and $k = 4$ varying at one time. We create 80 choice sets of two alternatives each. The resulting design is optimal under the assumption $\boldsymbol{\beta} = \mathbf{0}$ (Anderson 2003). The following steps create and evaluate the design:

```
%mktbibd(b=20, nattrs=16, setsize=4, seed=17)

%mktex(4 2 ** 4, n=8, seed=306)

proc sort data=randomized out=randes(drop=x1);
   by x2 x1;
   run;

proc print noobs data=randes; run;

%mktppro(design=randes, ibd=bibd)
```

```
%choiceff(data=chdes,              /* candidate set of choice sets        */
          init=chdes,              /* initial design                      */
          initvars=x1-x6,          /* factors in the initial design       */
          model=class(x1-x16 / sta),/* model with stdz orthogonal coding   */
          nsets=80,                /* number of choice sets               */
          nalts=2,                 /* number of alternatives              */
          rscale=                  /* relative D-efficiency scale factor  */
          %sysevalf(80 * 4 / 16),  /* 4 of 16 attrs in 80 sets vary       */
          beta=zero)               /* assumed beta vector, Ho: b=0        */

proc print data=chdes(obs=12); id set; by set; run;

%mktdups(generic, data=best, factors=x1-x16, nalts=2)
```

The first step makes the BIBD as we saw in the previous step. It tells us which attributes to vary in each choice set. The following design is a BIBD:

```
                    Balanced Incomplete Block Design

                          x1    x2    x3    x4

                           1     4    16    10
                          16    13    14    11
                          12    15     4    14
                           8    10     2    13
                          13     9     1    15
                           4     6    13     5
                          10     5    15     7
                           5    12     8    16
                           9    14     5     2
                           3     5    11     1
                           7    13     3    12
                          16     2     3    15
                          14     6    10     3
                           8     3     9     4
                           9    11    12    10
                           1     7    14     8
                           2     1    12     6
                          11     4     2     7
                           6    16     7     9
                          15     8     6    11
```

Next we need to know how those attributes are to vary. We need an OA for that. The OA must be a p^k subset of an array $p^k s^1$ in $p \times s$ runs with $k \le s$. Here we have $p = 2$ alternatives and we want to vary $k = 4$ at a time. The smallest array that works is $2^4 4^1$ in 8 runs. The %MktEx step generates this array, and the PROC PRINT step displays it. The results are as follows:

```
x2    x3    x4    x5

 1     1     2     1
 1     2     2     2
 1     1     1     2
 1     2     1     1
 2     2     1     2
 2     1     1     1
 2     2     2     1
 2     1     2     2
```

This array has two blocks of observations. The first block shows the attribute levels for the first alternative, and the second block shows the attribute levels for the second alternative.

There are several ways to create an OA that is sorted in the right way. The preceding %MktEx step requests the s-level factor first, then it sorts the randomized design by the first p-level factor followed by the s-level factor. Finally, it discards the s-level factor. Alternatively, you could first request one of the p-level factors, then request the s-level factor, then request the remaining $(k - 1)$ p-level factors. Then in the out=design data set, *after* the array is created, discard x2, the s-level factor. Note that you cannot drop the second factor in the %MktEx step by specifying out=design(drop=x2), because %MktEx will drop the variable and then sort, and your rows will be in the wrong order. The former approach has the advantage of being less likely to produce alternatives that are constant (that is, in one alternative, all attributes are absent and in other alternatives, all attributes are present).

The %MktPPro step combines the BIBD and the OA. The data set with the partial-profile choice design is called chdes. We can use the %ChoicEff macro to evaluate the design. The last part of the output is as follows:

```
              Final Results

Design                      1
Choice Sets                80
Alternatives                2
Parameters                 16
Maximum Parameters         80
D-Efficiency          20.0000
Relative D-Eff       100.0000
1 / Choice Sets        0.0125
```

```
          Variable                                      Standard
     n      Name      Label     Variance    DF      Error

     1      x11       x1 1        0.05       1     0.22361
     2      x21       x2 1        0.05       1     0.22361
     3      x31       x3 1        0.05       1     0.22361
     4      x41       x4 1        0.05       1     0.22361
     5      x51       x5 1        0.05       1     0.22361
     6      x61       x6 1        0.05       1     0.22361
     7      x71       x7 1        0.05       1     0.22361
     8      x81       x8 1        0.05       1     0.22361
     9      x91       x9 1        0.05       1     0.22361
    10      x101      x10 1       0.05       1     0.22361
    11      x111      x11 1       0.05       1     0.22361
    12      x121      x12 1       0.05       1     0.22361
    13      x131      x13 1       0.05       1     0.22361
    14      x141      x14 1       0.05       1     0.22361
    15      x151      x15 1       0.05       1     0.22361
    16      x161      x16 1       0.05       1     0.22361
                                            ==
                                            16
```

The variances are constant, and the relative *D*-efficiency is 100. *D*-efficiency is 20, which is 4 / 16 of the number of choice sets, 80. Since 25% of the attributes vary, *D*-efficiency is 25% of what we would expect in a full-profile generic choice design. With the `rscale=20` option, relative *D*-efficiency is based on the true maximum. The macro function `%sysevalf` is used to evaluate the expression `80 * 4 / 16` to get the 20. This function evaluates expressions that contain or produce floating point numbers and then returns the result. This particular expression could be evaluated with `%eval`, which does integer arithmetic, but that is not always the case. In summary, this design is optimal for partial profiles, and it is 25% efficient relative to an optimal generic design where all attributes can vary.

The final PROC PRINT step displays the first six choice sets as follows:

```
Set  x1  x2  x3  x4  x5  x6  x7  x8  x9  x10  x11  x12  x13  x14  x15  x16

 1    1   1   1   1   1   1   1   1   1   1    1    1    1    1    1    2
      2   1   1   2   1   1   1   1   1   2    1    1    1    1    1    1

 2    1   1   1   2   1   1   1   1   1   2    1    1    1    1    1    2
      2   1   1   1   1   1   1   1   1   1    1    1    1    1    1    1

 3    1   1   1   1   1   1   1   1   1   2    1    1    1    1    1    1
      2   1   1   2   1   1   1   1   1   1    1    1    1    1    1    2

 4    1   1   1   2   1   1   1   1   1   1    1    1    1    1    1    1
      2   1   1   1   1   1   1   1   1   2    1    1    1    1    1    2

 5    1   1   1   1   1   1   1   1   1   1    1    1    1    2    1    1
      1   1   1   1   1   1   1   1   1   1    2    1    2    1    1    2

 6    1   1   1   1   1   1   1   1   1   1    2    1    2    2    1    1
      1   1   1   1   1   1   1   1   1   1    1    1    1    1    1    2
```

There are a total of 80 choice sets, $b = 20$ blocks of size $s = 4$. The first four choice sets show the 8-run OA embedded in the partial-profile design in columns 1, 4, 10, and 16, as directed by the first row of the BIBD. The next four choice sets (only two are shown) show the same 8-run OA embedded in the partial-profile design in columns 11, 13, 14, and 16, as directed by the second row of the BIBD.

We have already seen that the choice design is statistically optimal. However, it might not be optimal from a practical usage point of view. In every choice set, there is a pair of alternatives, one with three attributes present and one with only one attribute present. This might not meet your needs (or maybe it does). You always need to inspect your designs to see how well they work for your purposes. Statistical efficiency and optimality are just one part of the picture. With the OA $4^1 2^4$ in 8 runs, there is one other kind of OA that you can get by specifying other random number seeds, that always produces a choice set (one in each block of choice sets) where all four attributes are present in one alternative and none are present in the other alternative. This might be worse. This kind of problem is less likely to be an issue when you vary more than four attributes.

The output from the `%MktDups` macro is as follows:

```
Design:          Generic
Factors:         x1-x16
                 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16
Sets w Dup Alts: 0
Duplicate Sets:  0
```

There are no duplicates.

80 Choice Sets, 16 Binary Attributes, Four Varying, Part 2

This example is identical to the previous example except that now $b = 12$ blocks are requested instead of 20, resulting in 48 choice sets instead of 80. The following steps create the design:

```
%mktbibd(b=12, nattrs=16, setsize=4, seed=17)

%mktex(4 2 ** 4, n=8, seed=306)

proc sort data=randomized out=randes(drop=x1);
   by x2 x1;
   run;

%mktppro(ibd=bibd, design=randes)

%choiceff(data=chdes,              /* candidate set of choice sets         */
          init=chdes,              /* initial design                       */
          initvars=x1-x6,          /* factors in the initial design        */
          model=class(x1-x16 / sta),/* model with stdz orthogonal coding   */
          nsets=48,                /* number of choice sets                */
          nalts=2,                 /* number of alternatives               */
          rscale=                  /* relative D-efficiency scale factor   */
          %sysevalf(48 * 4 / 16),  /* 4 of 16 attrs in 80 sets vary        */
          beta=zero)               /* assumed beta vector, Ho: b=0         */

%mktdups(generic, data=best, factors=x1-x16, nalts=2)
```

The first part of the output from the `%MktBIBD` macro is as follows:

```
                    Block Design Efficiency Criterion       98.0066
                    Number of Attributes, t                      16
                    Set Size, k                                   4
                    Number of Sets, b                            12
                    Average Attribute Frequency                   3
                    Average Pairwise Frequency                  0.6
                    Total Sample Size                            48
                    Positional Frequencies Optimized?           Yes
```

We can see that our results, an unbalanced block design, is not a BIBD since the efficiency is not 100. The attribute by attribute frequencies are as follows:

```
                  Attribute by Attribute Frequencies

          1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16

      1   3   1   1   0   0   0   1   1   0   1   1   1   1   0   0   1
      2       3   0   0   1   1   0   0   0   1   0   1   1   1   1   1
      3           3   1   1   1   1   1   0   0   0   1   0   0   1   1
      4               3   1   0   1   0   1   1   0   1   1   1   0   1
      5                   3   1   0   1   0   1   1   1   0   1   0   0
      6                       3   1   1   1   1   0   0   1   0   1   0
      7                           3   0   0   1   1   0   1   1   1   0
      8                               3   1   0   1   0   1   1   0   1
      9                                   3   1   1   1   1   0   1   1
     10                                       3   1   0   0   0   0   1
     11                                           3   1   0   1   1   0
     12                                               3   1   0   1   0
     13                                                   3   1   0   0
     14                                                       3   1   1
     15                                                           3   1
     16                                                               3
```

Each attribute occurs exactly 3 times, but each pair of attributes does not occur equally often. With 20 blocks, each pair occurred exactly once, so with fewer blocks, the best we can do is some zeros and some ones. The same is true for the attribute by position frequencies, which are as follows:

```
                  Attribute by Position Frequencies

                          1   2   3   4

                      1   1   1   1   0
                      2   1   0   1   1
                      3   1   1   1   0
                      4   1   0   1   1
                      5   1   1   1   0
                      6   0   1   1   1
                      7   1   1   0   1
                      8   1   1   0   1
                      9   1   0   1   1
                     10   1   0   1   1
                     11   1   1   1   0
                     12   0   1   1   1
                     13   0   1   1   1
                     14   0   1   1   1
                     15   1   1   0   1
```

The unbalanced block design is as follows:

```
                         Design

                    x1    x2    x3    x4

                     9    11    12    15
                    15     7     3     6
                     2    15    14    16
                     7     1    11    10
                     8    13     6     9
                     3     5     4    12
                    16     3     1     8
                    10    16     9     4
                     1    12     2    13
                     4    14    13     7
                    11     8     5    14
                     5     6    10     2
```

The last part of the output from the `%ChoicEff` macro's evaluation of the design is as follows:

```
                     Final Results

                Design                   1
                Choice Sets             48
                Alternatives             2
                Parameters              16
                Maximum Parameters      48
                D-Efficiency       12.0000
                Relative D-Eff    100.0000
                D-Error             0.0833
                1 / Choice Sets     0.0208
```

```
             Variable                                     Standard
        n      Name       Label    Variance     DF        Error

        1      x11        x1 1     0.083333      1       0.28868
        2      x21        x2 1     0.083333      1       0.28868
        3      x31        x3 1     0.083333      1       0.28868
        4      x41        x4 1     0.083333      1       0.28868
        5      x51        x5 1     0.083333      1       0.28868
        6      x61        x6 1     0.083333      1       0.28868
        7      x71        x7 1     0.083333      1       0.28868
        8      x81        x8 1     0.083333      1       0.28868
        9      x91        x9 1     0.083333      1       0.28868
       10      x101       x10 1    0.083333      1       0.28868
       11      x111       x11 1    0.083333      1       0.28868
       12      x121       x12 1    0.083333      1       0.28868
       13      x131       x13 1    0.083333      1       0.28868
       14      x141       x14 1    0.083333      1       0.28868
       15      x151       x15 1    0.083333      1       0.28868
       16      x161       x16 1    0.083333      1       0.28868
                                                ==
                                                16
```

The relative *D*-efficiency is 100% corresponding to 4 / 16 of 48 choice sets. This design is optimal! It is optimal even though our block design is not balanced. We might not have used a real BIBD, but we did use a real OA, and that is what is critical to achieve optimality. It is optimal for a partial-profile design in 48 choices sets even though it provides less information than a design optimized for a larger number of choice sets such as 80.

The output from the `%MktDups` macro is as follows:

```
 Design:          Generic
 Factors:         x1-x16
                  x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16
 Sets w Dup Alts: 0
 Duplicate Sets:  0
```

There are no duplicates.

Note again, however, that while this design is statistically optimal, it might not be optimal from a practical usage point of view. Besides the problems with the imbalance in the number of attributes shown in each alternative, now every attribute is no longer paired with every other attribute. These are issues you need to think about when you evaluate your designs.

80 Choice Sets, 16 Binary Attributes, Four Varying, Part 3

This example is identical to the previous example except that now $b = 8$ blocks are requested instead of 12 or 20, resulting in 32 choice sets instead of 48 or 80. The following steps create and evaluate the design:

```
%mktbibd(b=8, nattrs=16, setsize=4, seed=17)

%mktex(4 2 ** 4, n=8, seed=306)

proc sort data=randomized out=randes(drop=x1);
   by x2 x1;
   run;

%mktppro(ibd=bibd, design=randes)

%choiceff(data=chdes,              /* candidate set of choice sets         */
          init=chdes,              /* initial design                       */
          initvars=x1-x6,          /* factors in the initial design        */
          model=class(x1-x16 / sta),/* model with stdz orthogonal coding   */
          nsets=32,                /* number of choice sets                */
          nalts=2,                 /* number of alternatives               */
          rscale=                  /* relative D-efficiency scale factor   */
          %sysevalf(32 * 4 / 16),  /* 4 of 16 attrs in 32 sets vary        */
          beta=zero)               /* assumed beta vector, Ho: b=0         */

%mktdups(generic, data=best, factors=x1-x16, nalts=2)
```

The attribute by attribute frequencies are as follows:

```
                     Attribute by Attribute Frequencies

        1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16

   1    2  0  1  0  0  0  0  1  1  0  1  1  1  0  0  0
   2       2  1  0  0  0  1  0  0  1  0  0  1  0  1  1
   3          2  0  0  0  1  1  0  1  0  1  0  0  0  0
   4             2  0  1  1  0  1  0  0  1  0  1  0  1
   5                2  1  0  1  0  1  1  0  0  1  1  0
   6                   2  0  0  0  1  1  1  0  0  0  1
   7                      2  0  1  1  0  0  0  1  0  0
   8                         2  0  0  0  1  0  1  1  0
   9                            2  0  1  0  1  1  0  0
  10                               2  1  0  0  0  0  0
  11                                  2  0  1  0  0  0
  12                                     2  0  0  0  1
  13                                        2  0  1  1
  14                                           2  1  0
  15                                              2  1
  16                                                 2
```

Again, we have a constant on the diagonal and ones and zeros off diagonal. Again, this design is optimal. The results are as follows:

```
                 Final Results

           Design                     1
           Choice Sets               32
           Alternatives               2
           Parameters                16
           Maximum Parameters        32
           D-Efficiency          8.0000
           Relative D-Eff      100.0000
           D-Error               0.1250
           1 / Choice Sets       0.0313
```

```
            Variable                                          Standard
     n        Name       Label     Variance     DF        Error

     1        x11        x1 1         0.125      1      0.35355
     2        x21        x2 1         0.125      1      0.35355
     3        x31        x3 1         0.125      1      0.35355
     4        x41        x4 1         0.125      1      0.35355
     5        x51        x5 1         0.125      1      0.35355
     6        x61        x6 1         0.125      1      0.35355
     7        x71        x7 1         0.125      1      0.35355
     8        x81        x8 1         0.125      1      0.35355
     9        x91        x9 1         0.125      1      0.35355
    10        x101       x10 1        0.125      1      0.35355
    11        x111       x11 1        0.125      1      0.35355
    12        x121       x12 1        0.125      1      0.35355
    13        x131       x13 1        0.125      1      0.35355
    14        x141       x14 1        0.125      1      0.35355
    15        x151       x15 1        0.125      1      0.35355
    16        x161       x16 1        0.125      1      0.35355
                                                ==
                                                16
```

It is optimal for a partial-profile design in 32 choices sets even though it provides less information than designs optimized for a larger number of choice sets such as 48 or 80.

The output from the %MktDups macro is as follows:

```
Design:          Generic
Factors:         x1-x16
                 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16
Sets w Dup Alts: 0
Duplicate Sets:  0
```

There are no duplicates.

80 Choice Sets, 16 Binary Attributes, Four Varying, Part 4

This example is identical to the previous example except that now $b = 7$ blocks are requested instead of a multiple of 4. The following steps create the design:

```
%mktbibd(b=7, nattrs=16, setsize=4, seed=17)

%mktex(4 2 ** 4, n=8, seed=306)

proc sort data=randomized out=randes(drop=x1);
   by x2 x1;
   run;
```

```
%mktppro(ibd=bibd, design=randes)

%choiceff(data=chdes,              /* candidate set of choice sets        */
          init=chdes,              /* initial design                      */
          initvars=x1-x6,          /* factors in the initial design       */
          model=class(x1-x16 / sta),/* model with stdz orthogonal coding   */
          nsets=28,                /* number of choice sets               */
          nalts=2,                 /* number of alternatives              */
          rscale=                  /* relative D-efficiency scale factor  */
          %sysevalf(28 * 4 / 16),  /* 4 of 16 attrs in 28 sets vary       */
          beta=zero)               /* assumed beta vector, Ho: b=0        */

%mktdups(generic, data=best, factors=x1-x16, nalts=2)
```

The attribute by attribute frequencies are as follows:

```
                    Attribute by Attribute Frequencies

         1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16

     1   2  0  0  1  0  0  0  0  1  0  1  1  1  0  1  0
     2      2  0  0  0  0  1  1  0  1  0  0  1  0  1  1
     3         2  0  0  1  1  0  1  0  0  1  0  1  0  1
     4            1  0  0  0  0  0  0  0  1  0  0  1  0
     5               1  1  0  0  0  1  1  0  0  0  0  0
     6                  2  0  0  0  1  1  1  0  0  0  1
     7                     2  0  1  1  0  0  0  1  1  0
     8                        1  0  0  0  0  1  0  0  1
     9                           2  0  1  0  1  1  0  0
    10                              2  1  0  0  0  1  0
    11                                 2  0  1  0  0  0
    12                                    2  0  0  1  1
    13                                       2  0  0  1
    14                                          1  0  0
    15                                             2  0
    16                                                2
```

The 16 attributes cannot be equally distributed across the 7×4 entries in the incomplete block design, so the diagonal has nonconstant frequencies. The evaluation is as follows:

```
                              Final Results

                        Design                  1
                        Choice Sets            28
                        Alternatives            2
                        Parameters             16
                        Maximum Parameters     28
                        D-Efficiency       6.7272
                        Relative D-Eff    96.1024
                        D-Error            0.1487
                        1 / Choice Sets    0.0357

                Variable                                     Standard
          n       Name      Label    Variance    DF       Error

          1      x11       x1 1       0.125       1     0.35355
          2      x21       x2 1       0.125       1     0.35355
          3      x31       x3 1       0.125       1     0.35355
          4      x41       x4 1       0.250       1     0.50000
          5      x51       x5 1       0.250       1     0.50000
          6      x61       x6 1       0.125       1     0.35355
          7      x71       x7 1       0.125       1     0.35355
          8      x81       x8 1       0.250       1     0.50000
          9      x91       x9 1       0.125       1     0.35355
         10      x101      x10 1      0.125       1     0.35355
         11      x111      x11 1      0.125       1     0.35355
         12      x121      x12 1      0.125       1     0.35355
         13      x131      x13 1      0.125       1     0.35355
         14      x141      x14 1      0.250       1     0.50000
         15      x151      x15 1      0.125       1     0.35355
         16      x161      x16 1      0.125       1     0.35355
                                                 ==
                                                 16
```

The design does not have a relative *D*-efficiency of 100%, so it is not optimal relative to a hypothetical, optimal partial-profile design. With seven blocks, or any *b* not a multiple of 4, the kind of combinatorial optimality we saw previously is not possible.

The output from the `%MktDups` macro is as follows:

```
  Design:          Generic
  Factors:         x1-x16
                   x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16
  Sets w Dup Alts: 0
  Duplicate Sets:  0
```

There are no duplicates.

Partial Profiles, the General Combinatorial Approach

In the previous sections, we saw several examples of using the `%MktBIBD` macro to make BIBDs, the `%MktEx` macro to make OAs, the `%MktPPro` macro to combine them into a partial-profile design, and the `%ChoicEff` macro to evaluate the design. Next an ad hoc wrapper macro that combines them into a single macro for making partial-profile designs based on this combinatorial structure is shown. It works for *attr* attributes and *lev*-level factors, with *vary* varying, and *maxv* × *lev* rows in each of the *b* blocks of choice sets. The macro is as follows:

```
%macro pp(                          /*-------------------------------------*/
       lev=,                        /* levels                              */
       nattrs=,                     /* total attributes                    */
       setsize=,                    /* number of attributes that vary      */
       maxv=,                       /* max number of attrs that can vary   */
       b=,                          /* number of blocks                    */
       seed=);                      /* random number seed                  */
                                    /*-------------------------------------*/

%let sets = %sysevalf(&b * &maxv);
%put NOTE: nattrs=&nattrs, setsize=&setsize, sets=&sets, b=&b..;

%mktbibd(b=&b, nattrs=&nattrs, setsize=&setsize, seed=&seed, positer=0)

%mktex(&maxv &lev ** &setsize, n=&lev * &maxv)

proc sort data=randomized out=randes(drop=x1);
   by x2 x1;
   run;

%mktppro(ibd=bibd, design=randes)

%choiceff(data=chdes,               /* candidate set of choice sets        */
          init=chdes,               /* initial design                      */
          initvars=x1-x&nattrs,     /* factors in the initial design       */
          intiter=0,                /* evaluate without internal iterations */
          model=class(x1-x&nattrs / sta),/* model with stdz orthog coding  */
          nsets=&sets,              /* number of choice sets               */
          nalts=&lev,               /* number of alternatives              */
          rscale=                   /* relative D-efficiency scale factor  */
          %sysevalf(&sets * &setsize / &nattrs),
          beta=zero)                /* assumed beta vector, Ho: b=0        */

proc print data=chdes; by set; id set; var x:; run;

%mktdups(generic, data=best, factors=x1-x&nattrs, nalts=&lev)
%mend;
```

The next steps create 25 examples of partial-profile designs. Thousands more could be made. You can use any positive integer $< 2^{31} - 1$ for a seed.* Note that some are subject to problems with duplicates. The following steps create the designs:

```
%pp(lev = 2, nattrs = 16, setsize =  4, maxv =  4, b =  20, seed =  17)
%pp(lev = 4, nattrs = 16, setsize =  4, maxv =  4, b =  20, seed =  93)
%pp(lev = 2, nattrs = 16, setsize =  8, maxv =  8, b =  30, seed = 109)
%pp(lev = 3, nattrs = 13, setsize =  6, maxv =  6, b =  26, seed = 114)
%pp(lev = 2, nattrs = 23, setsize = 12, maxv = 12, b =  23, seed = 121)
%pp(lev = 5, nattrs = 11, setsize =  5, maxv =  5, b =  11, seed = 145)
%pp(lev = 3, nattrs = 19, setsize =  9, maxv =  9, b =  19, seed = 151)
%pp(lev = 2, nattrs = 22, setsize =  7, maxv = 16, b =  22, seed = 205)
%pp(lev = 4, nattrs = 13, setsize =  3, maxv =  8, b =  26, seed = 238)
%pp(lev = 3, nattrs = 21, setsize =  5, maxv = 12, b =  21, seed = 289)
%pp(lev = 2, nattrs = 22, setsize =  8, maxv = 20, b =  11, seed = 292)
%pp(lev = 3, nattrs = 25, setsize =  9, maxv = 15, b =  25, seed = 306)
%pp(lev = 4, nattrs = 21, setsize =  5, maxv = 12, b =  21, seed = 350)
%pp(lev = 7, nattrs = 15, setsize =  7, maxv =  7, b =  15, seed = 368)
%pp(lev = 5, nattrs = 16, setsize = 10, maxv = 10, b =  16, seed = 377)
%pp(lev = 4, nattrs = 16, setsize =  8, maxv = 16, b =  10, seed = 382)
%pp(lev = 8, nattrs = 15, setsize =  8, maxv =  8, b =  15, seed = 396)
%pp(lev = 2, nattrs =  8, setsize =  2, maxv = 28, b =  28, seed = 420)
%pp(lev = 5, nattrs =  7, setsize =  6, maxv = 15, b =   7, seed = 424)
%pp(lev = 4, nattrs = 11, setsize =  6, maxv = 20, b =  11, seed = 448)
%pp(lev = 3, nattrs = 45, setsize = 12, maxv = 27, b =  45, seed = 462)
%pp(lev = 9, nattrs = 16, setsize =  4, maxv =  9, b =  20, seed = 472)
%pp(lev = 3, nattrs = 15, setsize =  5, maxv = 30, b =  21, seed = 495)
%pp(lev = 7, nattrs =  9, setsize =  3, maxv = 14, b =  12, seed = 513)
%pp(lev = 5, nattrs = 13, setsize =  4, maxv = 20, b =  13, seed = 522)
```

It is instructive to see how these parameters are set, particularly `maxv=` and `rep` parameters. In the first example, there are 16 two-level attributes, and four vary at a time. This corresponds to a BIBD with $t = 14$ and $k = 4$. We can run the `%MktBSize` macro with these specifications as follows:

```
%mktbsize(nattrs=16, setsize=4)
```

We see that a BIBD is available with each attribute appearing $r = 5$ times. This is the `rep=` parameter in the ad hoc macro. Remember though, that we do not need a BIBD in order to make good partial-profile designs. We can use other specifications as well, as long as there is an OA that works with our block design. We can find all OAs $p^k s^1$ in $p \times s$ runs with $k \leq s$ as follows:

*These are selected from some particularly interesting OA sections.

```
%mktorth(options=parent, maxlev=144)

data x(keep=n design);
   set mktdeslev;
   array x[144];
   c = 0; one = 0; k = 0;
   do i = 1 to 144;
      c + (x[i] > 0); /* how many differing numbers of levels */
      if x[i] > 1 then do; p = i; k = x[i]; end; /* p^k */
      if x[i] = 1 then do; one + 1; s = i;  end; /* s^1 */
      end;
   if c = 1 then do; c = 2; one = 1; s = p; k = p - 1; end;
   if c = 2 and one = 1 and k > 2 and s * p = n;
   design = compbl(left(design));
   run;

proc print noobs; by n; id n; run;
```

A few of the smaller ones that work are as follows:

```
                       n    Design

                       8    2 ** 4 4 ** 1

                      16    2 ** 8 8 ** 1
                            4 ** 5

                      18    3 ** 6 6 ** 1

                      24    2 ** 12 12 ** 1

                      25    5 ** 6

                      27    3 ** 9 9 ** 1

                      32    2 ** 16 16 ** 1
                            4 ** 8 8 ** 1

                      36    3 ** 12 12 ** 1

                      40    2 ** 20 20 ** 1

                      45    3 ** 9 15 ** 1

                      48    2 ** 24 24 ** 1
                            4 ** 12 12 ** 1

                      49    7 ** 8

                      50    5 ** 10 10 ** 1
```

The first example used $2^4 4^1$ in 8 runs ($2^k s^1$ or $2^{setsize} maxv^1$). Note that you can always select a subset of the available columns to vary ($vary < k$), and many of the examples do precisely that.

Efficient but Nonoptimal Partial Profiles

We can use the same methods to make efficient but nonoptimal partial-profile designs. With 12 attributes, six varying, and six-level factors, an OA is not possible in 36 runs. Furthermore, with 12 blocks of choice sets and 12 attributes and six varying, a BIBD is not possible either. Still, we can use an efficient factorial design and an unbalanced block design to make a partial-profile design as follows:

```
%mktbibd(b=12, nattrs=12, setsize=6, seed=93, positer=)

%mktex(6 ** 7, n=36, seed=238)

proc sort data=randomized out=randes(drop=x1);
   by x2 x1;
   run;

%mktppro(ibd=bibd, design=randes)

%choiceff(data=chdes,               /* candidate set of choice sets         */
          init=chdes,               /* initial design                       */
          initvars=x1-x12,          /* factors in the initial design        */
          intiter=0,                /* evaluate without internal iterations */
          model=class(x1-x12 / sta),/* model with stdz orthogonal coding    */
          nsets=72,                 /* number of choice sets                */
          nalts=6,                  /* number of alternatives               */
          rscale=                   /* relative D-efficiency scale factor   */
          %sysevalf(72 * 6 / 12),   /* 6 of 12 attrs in 72 sets vary        */
          beta=zero)                /* assumed beta vector, Ho: b=0         */

proc print; by set; id set; var x1-x12; run;

%mktdups(generic, data=best, factors=x1-x12, nalts=6)
```

Conclusions

This very long chapter has several purposes. It provides an introduction to the multinomial logit model used in choice modeling, it discusses linear model designs created by the `%MktEx` macro and their use in choice modeling, and it discusses using the `%ChoicEff` macro to find and evaluate designs. These two macros, along with all of the other macros, provide powerful tools for designing choice experiments in a variety of ways. Numerous other macros and techniques are discussed as well.

Multinomial Logit Models

Ying So

Warren F. Kuhfeld

Abstract

Multinomial logit models are used to model relationships between a polytomous response variable and a set of regressor variables. The term "multinomial logit model" includes, in a broad sense, a variety of models. The cumulative logit model is used when the response of an individual unit is restricted to one of a finite number of ordinal values. Generalized logit and conditional logit models are used to model consumer choices. This article focuses on the statistical techniques for analyzing discrete choice data and discusses fitting these models using SAS/STAT software.[*]

Introduction

Multinomial logit models are used to model relationships between a polytomous response variable and a set of regressor variables. These polytomous response models can be classified into two distinct types, depending on whether the response variable has an ordered or unordered structure.

In an ordered model, the response Y of an individual unit is restricted to one of m ordered values. For example, the severity of a medical condition may be: none, mild, and severe. The cumulative logit model assumes that the ordinal nature of the observed response is due to methodological limitations in collecting the data that results in lumping together values of an otherwise continuous response variable (McKelvey and Zavoina 1975). Suppose Y takes values $y_1, y_2, \ldots, y_m$ on some scale, where $y_1 < y_2 < \ldots < y_m$. It is assumed that the observable variable is a categorized version of a continuous latent variable U such that

$$Y = y_i \Leftrightarrow \alpha_{i-1} < U \leq \alpha_i, i = 1, \ldots, m$$

where $-\infty = \alpha_0 < \alpha_1 < \ldots < \alpha_m = \infty$. It is further assumed that the latent variable U is determined by the explanatory variable vector $\mathbf{x}$ in the linear form $U = -\boldsymbol{\beta}'\mathbf{x} + \epsilon$, where $\boldsymbol{\beta}$ is a vector of regression coefficients and ϵ is a random variable with a distribution function F. It follows that

$$\Pr\{Y \leq y_i | \mathbf{x}\} = F(\alpha_i + \boldsymbol{\beta}'\mathbf{x})$$

If F is the logistic distribution function, the cumulative model is also known as the proportional odds model. You can use PROC LOGISTIC or PROC PROBIT directly to fit the cumulative logit models. Although the cumulative model is the most widely used model for ordinal response data, other useful models include the adjacent-categories logit model and the continuation-ratio model (Agresti 1990).

[*]This chapter was presented at SUGI 20 by Ying So and can also be found in the SUGI 20 proceedings. Copies of this article (MR-2010G), the other chapters, sample code, and all of the macros are available on the Web `http://support.sas.com/resources/papers/tnote/tnote_marketresearch.html`.

In an unordered model, the polytomous response variable does not have an ordered structure. Two classes of models, the generalized logit models and the conditional logit models, can be used with nominal response data. The generalized logit model consists of a combination of several binary logits estimated simultaneously. For example, the response variable of interest is the occurrence or nonoccurrence of infection after a Caesarean section with two types of (I,II) infection. Two binary logits are considered: one for type I infection versus no infection and the other for type II infection versus no infection. The conditional logit model has been used in biomedical research to estimate relative risks in matched case-control studies. The nuisance parameters that correspond to the matched sets in an unconditional analysis are eliminated by using a conditional likelihood that contains only the relative risk parameters (Breslow and Day 1980). The conditional logit model was also introduced by McFadden (1974) in the context of econometrics.

In studying consumer behavior, an individual is presented with a set of alternatives and asked to choose the most preferred alternative. Both the generalized logit and conditional logit models are used in the analysis of discrete choice data. In a conditional logit model, a choice among alternatives is treated as a function of the characteristics of the alternatives, whereas in a generalized logit model, the choice is a function of the characteristics of the individual making the choice. In many situations, a mixed model that includes both the characteristics of the alternatives and the individual is needed for investigating consumer choice.

Consider an example of travel demand. People are asked to choose between travel by auto, plane or public transit (bus or train). The following SAS statements create the data set TRAVEL. The variables `AutoTime`, `PlanTime`, and `TranTime` represent the total travel time required to get to a destination by using auto, plane, or transit, respectively. The variable `Age` represents the age of the individual being surveyed, and the variable `Chosen` contains the individual's choice of travel mode.

```
data travel;
   input AutoTime PlanTime TranTime Age Chosen $;
   datalines;
10.0      4.5     10.5    32  Plane
 5.5      4.0      7.5    13  Auto
 4.5      6.0      5.5    41  Transit
 3.5      2.0      5.0    41  Transit
 1.5      4.5      4.0    47  Auto
10.5      3.0     10.5    24  Plane
 7.0      3.0      9.0    27  Auto
 9.0      3.5      9.0    21  Plane
 4.0      5.0      5.5    23  Auto
22.0      4.5     22.5    30  Plane
 7.5      5.5     10.0    58  Plane
11.5      3.5     11.5    36  Transit
 3.5      4.5      4.5    43  Auto
12.0      3.0     11.0    33  Plane
18.0      5.5     20.0    30  Plane
23.0      5.5     21.5    28  Plane
 4.0      3.0      4.5    44  Plane
 5.0      2.5      7.0    37  Transit
 3.5      2.0      7.0    45  Auto
```

```
   12.5      3.5      15.5    35  Plane
    1.5      4.0       2.0    22  Auto
   ;
```

In this example, `AutoTime`, `PlanTime`, and `TranTime` are alternative-specific variables, whereas `Age` is a characteristic of the individual. You use a generalized logit model to investigate the relationship between the choice of transportation and `Age`, and you use a conditional logit model to investigate how travel time affects the choice. To study how the choice depends on both the travel time and age of the individual, you need to use a mixed model that incorporates both types of variables.

A survey of the literature reveals a confusion in the terminology for the nominal response models. The term "multinomial logit model" is often used to describe the generalized logit model. The mixed logit is sometimes referred to as the multinomial logit model in which the generalized logit and the conditional logit models are special cases.

The following sections describe discrete choice models, illustrate how to use SAS/STAT software to fit these models, and discuss cross-alternative effects.

Modeling Discrete Choice Data

Consider an individual choosing among m alternatives in a choice set. Let Π_{jk} denote the probability that individual j chooses alternative k, let $\mathbf{X}_j$ represent the characteristics of individual j, and let $\mathbf{Z}_{jk}$ be the characteristics of the kth alternative for individual j. For example, $\mathbf{X}_j$ may be an age and each $\mathbf{Z}_{jk}$ a travel time.

The generalized logit model focuses on the individual as the unit of analysis and uses individual characteristics as explanatory variables. The explanatory variables, being characteristics of an individual, are constant over the alternatives. For example, for each of the m travel modes, $\mathbf{X}_j = (1 \; age)'$, and for the first subject, $\mathbf{X}_1 = (1 \; 32)'$. The probability that individual j chooses alternative k is

$$\Pi_{jk} = \frac{\exp(\boldsymbol{\beta}_k' \mathbf{X}_j)}{\sum_{l=1}^{m} \exp(\boldsymbol{\beta}_l' \mathbf{X}_j)} = \frac{1}{\sum_{l=1}^{m} \exp[(\boldsymbol{\beta}_l - \boldsymbol{\beta}_k)' \mathbf{X}_j)]}$$

$\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_m$ are m vectors of unknown regression parameters (each of which is different, even though $\mathbf{X}_j$ is constant across alternatives). Since $\sum_{k=1}^{m} \Pi_{jk} = 1$, the m sets of parameters are not unique. By setting the last set of coefficients to null (that is, $\boldsymbol{\beta}_m = \mathbf{0}$), the coefficients $\boldsymbol{\beta}_k$ represent the effects of the $\mathbf{X}$ variables on the probability of choosing the kth alternative over the last alternative. In fitting such a model, you estimate $m - 1$ sets of regression coefficients.

In the conditional logit model, the explanatory variables $\mathbf{Z}$ assume different values for each alternative and the impact of a unit of $\mathbf{Z}$ is assumed to be constant across alternatives. For example, for each of the m travel modes, $\mathbf{Z}_{jk} = (time)'$, and for the first subject, $\mathbf{Z}_{11} = (10)'$, $\mathbf{Z}_{12} = (4.5)'$, and $\mathbf{Z}_{13} = (10.5)'$. The probability that the individual j chooses alternative k is

$$\Pi_{jk} = \frac{\exp(\boldsymbol{\theta}' \mathbf{Z}_{jk})}{\sum_{l=1}^{m} \exp(\boldsymbol{\theta}' \mathbf{Z}_{jl})} = \frac{1}{\sum_{l=1}^{m} \exp[\boldsymbol{\theta}' (\mathbf{Z}_{jl} - \mathbf{Z}_{jk})]}$$

$\boldsymbol{\theta}$ is a single vector of regression coefficients. The impact of a variable on the choice probabilities derives from the difference of its values across the alternatives.

For the mixed logit model that includes both characteristics of the individual and the alternatives, the choice probabilities are

$$\Pi_{jk} = \frac{\exp(\boldsymbol{\beta}_k'\mathbf{X}_j + \boldsymbol{\theta}'\mathbf{Z}_{jk})}{\sum_{l=1}^{m}\exp(\boldsymbol{\beta}_l'\mathbf{X}_j + \boldsymbol{\theta}'\mathbf{Z}_{jl})}$$

$\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_{m-1}$ and $\boldsymbol{\beta}_m \equiv \mathbf{0}$ are the alternative-specific coefficients, and $\boldsymbol{\theta}$ is the set of global coefficients.

Fitting Discrete Choice Models

The CATMOD procedure in SAS/STAT software directly fits the generalized logit model. SAS/STAT software does not yet have a procedure that is specially designed to fit the conditional or mixed logit models. However, with some preliminary data processing, you can use the PHREG procedure to fit these models.

The PHREG procedure fits the Cox proportional hazards model to survival data (see the SAS/STAT documentation). The partial likelihood of Breslow has the same form as the likelihood in a conditional logit model.

Let z_l denote the vector of explanatory variables for individual l. Let $t_1 < t_2 < \ldots < t_k$ denote k distinct ordered event times. Let d_i denote the number of failures at t_i. Let s_i be the sum of the vectors z_l for those individuals that fail at t_i, and let $\mathcal{R}_i$ denote the set of indices for those who are at risk just before t_i.

The Breslow (partial) likelihood is

$$L_B(\boldsymbol{\theta}) = \prod_{i=1}^{k} \frac{\exp(\boldsymbol{\theta}' s_i)}{[\sum_{l\in\mathcal{R}_i}\exp(\boldsymbol{\theta}' z_l)]^{d_i}}$$

In a stratified analysis, the partial likelihood is the product of the partial likelihood for each individual stratum. For example, in a study of the time to first infection from a surgery, the variables of a patient consist of `Time` (time from surgery to the first infection), `Status` (an indicator of whether the observation time is censored, with value 2 identifying a censored time), `Z1` and `Z2` (explanatory variables thought to be related to the time to infection), and `Grp` (a variable identifying the stratum to which the observation belongs). The specification in PROC PHREG for fitting the Cox model using the Breslow likelihood is as follows:

```
proc phreg;
   model time*status(2) = z1 z2 / ties=breslow;
   strata grp;
   run;
```

To cast the likelihood of the conditional logit model in the form of the Breslow likelihood, consider m artificial observed times for each individual who chooses one of m alternatives. The kth alternative is chosen at time 1; the choices of all other alternatives (second choice, third choice, ...) are not observed and would have been chosen at some later time. So a choice variable is coded with an observed time value of 1 for the chosen alternative and a larger value, 2, for all unchosen (unobserved or censored alternatives). For each individual, there is exactly one event time (1) and $m-1$ nonevent times, and the risk set just prior to this event time consists of all the m alternatives. For individual j with alternative-specific characteristics $\mathbf{Z}_{jl}$, the Breslow likelihood is then

$$L_B(\boldsymbol{\theta}) = \frac{\exp(\boldsymbol{\theta}'\mathbf{Z}_{jk})}{\sum_{l=1}^{m} \exp(\boldsymbol{\theta}'\mathbf{Z}_{jl})}$$

This is precisely the probability that individual j chooses alternative k in a conditional logit model. By stratifying on individuals, you get the likelihood of the conditional logit model. Note that the observed time values of 1 and 2 are chosen for convenience; however, the censored times have to be larger than the event time to form the correct risk set.

Before you invoke PROC PHREG to fit the conditional logit, you must arrange your data in such a way that there is a survival time for each individual-alternative. In the example of travel demand, let `Subject` identify the individuals, let `TravTime` represent the travel time for each mode of transportation, and let `Choice` have a value 1 if the alternative is chosen and 2 otherwise. The `Choice` variable is used as the artificial time variable as well as a censoring variable in PROC PHREG. The following SAS statements reshape the data set TRAVEL into data set CHOICE and display the first nine observations:

```
data choice(keep=subject mode travtime choice);
   array times[3] autotime plantime trantime;
   array allmodes[3] $ _temporary_ ('Auto' 'Plane' 'Transit');
   set travel;
   Subject = _n_;
   do i = 1 to 3;
      Mode = allmodes[i];
      TravTime = times[i];
      Choice = 2 - (chosen eq mode);
      output;
   end;
   run;

proc print data=choice(obs=9);
   run;
```

```
                                  Trav
Obs    Subject    Mode            Time    Choice

  1       1       Auto            10.0       2
  2       1       Plane            4.5       1
  3       1       Transit         10.5       2
  4       2       Auto             5.5       1
  5       2       Plane            4.0       2
  6       2       Transit          7.5       2
  7       3       Auto             4.5       2
  8       3       Plane            6.0       2
  9       3       Transit          5.5       1
```

Notice that each observation in TRAVEL corresponds to a block of three observations in CHOICE, exactly one of which is chosen.

The following SAS statements invoke PROC PHREG to fit the conditional logit model. The Breslow likelihood is requested by specifying `ties=breslow`. `Choice` is the artificial time variable, and a value of 2 identifies censored times. `Subject` is used as a stratification variable.

```
proc phreg data=choice;
   model choice*choice(2) = travtime / ties=breslow;
   strata subject;
   title 'Conditional Logit Model Using PHREG';
   run;
```

```
                         Conditional Logit Model Using PHREG

                                 The PHREG Procedure

                       Analysis of Maximum Likelihood Estimates

                    Parameter     Standard                               Hazard
    Variable   DF    Estimate        Error   Chi-Square   Pr > ChiSq      Ratio

    TravTime    1    -0.26549      0.10215       6.7551       0.0093      0.767
```

To study the relationship between the choice of transportation and the age of people making the choice, the analysis is based on the generalized logit model. You can use PROC CATMOD directly to fit the generalized logit model (see the *SAS/STAT User's Guide*). In the following invocation of PROC CATMOD, `Chosen` is the response variable and `Age` is the explanatory variable:

```
proc catmod data=travel;
   direct age;
   model chosen=age;
   title 'Multinomial Logit Model Using Catmod';
   run;
```

```
                       Response Profiles

                      Response    Chosen
                      ------------------
                          1       Auto
                          2       Plane
                          3       Transit


              Analysis of Maximum Likelihood Estimates

              Function                 Standard          Chi-
 Parameter    Number      Estimate        Error        Square    Pr > ChiSq
 ---------------------------------------------------------------------------
 Intercept       1          3.0449       2.4268          1.57        0.2096
                 2          2.7212       2.2929          1.41        0.2353
 Age             1         -0.0710       0.0652          1.19        0.2762
                 2         -0.0500       0.0596          0.70        0.4013
```

Note that there are two intercept coefficients and two slope coefficients for `Age`. The first `Intercept` and the first `Age` coefficients correspond to the effect on the probability of choosing auto over transit, and the second intercept and second age coefficients correspond to the effect of choosing plane over transit.

Let $\mathbf{X}_j$ be a $(p+1)$-vector representing the characteristics of individual j. The generalized logit model can be cast in the framework of a conditional model by defining the global parameter vector $\boldsymbol{\theta}$ and the alternative-specific regressor variables $\mathbf{Z}_{jk}$ as follows:

$$\boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \\ \vdots \\ \boldsymbol{\beta}_{m-1} \end{bmatrix} \quad \mathbf{Z}_{j1} = \begin{bmatrix} \mathbf{X}_j \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix} \quad \mathbf{Z}_{j2} = \begin{bmatrix} \mathbf{0} \\ \mathbf{X}_j \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix} \quad \cdots \quad \mathbf{Z}_{j,m-1} = \begin{bmatrix} \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \mathbf{X}_j \end{bmatrix} \quad \mathbf{Z}_{jm} = \begin{bmatrix} \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}$$

where the $\mathbf{0}$ is a $(p+1)$-vector of zeros. The probability that individual j chooses alternative k for the generalized logit model is put in the form that corresponds to a conditional logit model as follows:

$$\begin{aligned} \Pi_{jk} &= \frac{\exp(\boldsymbol{\beta}_k'\mathbf{X}_j)}{\sum_{l=1}^m \exp(\boldsymbol{\beta}_l'\mathbf{X}_j)} \\ &= \frac{\exp(\boldsymbol{\theta}'\mathbf{Z}_{jk})}{\sum_{l=1}^m \exp(\boldsymbol{\theta}'\mathbf{Z}_{jl})} \end{aligned}$$

Here, the vector $\mathbf{X}_j$ representing the characteristics of individual j includes the element 1 for the intercept parameter (provided that the intercept parameters are to be included in the model).

By casting the generalized logit model into a conditional logit model, you can then use PROC PHREG to analyze the generalized logit model. In the example of travel demand, the alternative-specific variables `Auto`, `Plane`, `AgeAuto`, and `AgePlane` are created from the individual characteristic variable `Age`. The following SAS statements reshape the data set TRAVEL into data set CHOICE2 and display the first nine observations:

```
data choice2;
   array times[3] autotime plantime trantime;
   array allmodes[3] $ _temporary_ ('Auto' 'Plane' 'Transit');
   set travel;
   Subject = _n_;
   do i = 1 to 3;
      Mode = allmodes[i];
      TravTime = times[i];
      Choice = 2 - (chosen eq mode);
      Auto = (i eq 1);
      Plane = (i eq 2);
      AgeAuto = auto * age;
      AgePlane = plane * age;
      output;
   end;
   keep subject mode travtime choice auto plane ageauto ageplane;
   run;

proc print data=choice2(obs=9);
   run;
```

```
                               Trav                                    Age      Age
  Obs     Subject     Mode      Time     Choice     Auto     Plane     Auto     Plane

    1        1        Auto      10.0        2         1        0        32         0
    2        1        Plane      4.5        1         0        1         0        32
    3        1        Transit   10.5        2         0        0         0         0
    4        2        Auto       5.5        1         1        0        13         0
    5        2        Plane      4.0        2         0        1         0        13
    6        2        Transit    7.5        2         0        0         0         0
    7        3        Auto       4.5        2         1        0        41         0
    8        3        Plane      6.0        2         0        1         0        41
    9        3        Transit    5.5        1         0        0         0         0
```

The following SAS statements invoke PROC PHREG to fit the generalized logit model:

```
proc phreg data=choice2;
   model choice*choice(2) = auto plane ageauto ageplane /
         ties=breslow;
   strata subject;
   title 'Generalized Logit Model Using PHREG';
   run;
```

```
                     Generalized Logit Model Using PHREG

                            The PHREG Procedure

                   Analysis of Maximum Likelihood Estimates

                  Parameter     Standard                                 Hazard
 Variable   DF     Estimate        Error   Chi-Square   Pr > ChiSq        Ratio

 Auto        1      3.04494      2.42682       1.5743       0.2096       21.009
 Plane       1      2.72121      2.29289       1.4085       0.2353       15.199
 AgeAuto     1     -0.07097      0.06517       1.1859       0.2762        0.931
 AgePlane    1     -0.05000      0.05958       0.7045       0.4013        0.951
```

By transforming individual characteristics into alternative-specific variables, the mixed logit model can be analyzed as a conditional logit model.

Analyzing the travel demand data for the effects of both travel time and age of individual requires the same data set as the generalized logit model, only now the `TravTime` variable will be used as well. The following SAS statements use PROC PHREG to fit the mixed logit model:

```
proc phreg data=choice2;
   model choice*choice(2) = auto plane ageauto ageplane travtime /
         ties=breslow;
   strata subject;
   title 'Mixed Logit Model Using PHREG';
   run;
```

```
                        Mixed Logit Model Using PHREG

                            The PHREG Procedure

                   Analysis of Maximum Likelihood Estimates

                  Parameter     Standard                                 Hazard
 Variable   DF     Estimate        Error   Chi-Square   Pr > ChiSq        Ratio

 Auto        1      2.50069      2.39585       1.0894       0.2966       12.191
 Plane       1     -2.77912      3.52929       0.6201       0.4310        0.062
 AgeAuto     1     -0.07826      0.06332       1.5274       0.2165        0.925
 AgePlane    1      0.01695      0.07439       0.0519       0.8198        1.017
 TravTime    1     -0.60845      0.27126       5.0315       0.0249        0.544
```

A special case of the mixed logit model is the conditional logit model with alternative-specific constants. Each alternative in the model can be represented by its own intercept, which captures the unmeasured desirability of the alternative.

```
proc phreg data=choice2;
   model choice*choice(2) = auto plane travtime / ties=breslow;
   strata subject;
   title 'Conditional Logit Model with Alternative Specific Constants';
   run;
```

```
            Conditional Logit Model with Alternative Specific Constants

                                 The PHREG Procedure

                         Analysis of Maximum Likelihood Estimates

                    Parameter    Standard                                 Hazard
    Variable   DF    Estimate       Error   Chi-Square   Pr > ChiSq        Ratio

    Auto        1    -0.11966     0.70820       0.0285       0.8658        0.887
    Plane       1    -1.63145     1.24251       1.7241       0.1892        0.196
    TravTime    1    -0.48665     0.20725       5.5139       0.0189        0.615
```

With transit as the reference mode, the intercept for auto, which is negative, may reflect the inconvenience of having to drive over traveling by bus/train, and the intercept for plane may reflect the high expense of traveling by plane over bus/train.

Cross-Alternative Effects

Discrete choice models are often derived from the principle of maximum random utility. It is assumed that an unobserved utility V_k is associated with the kth alternative, and the response function Y is determined by

$$Y = k \Leftrightarrow V_k = \max\{V_l, 1 \leq l \leq m\}$$

Both the generalized logit and the conditional logit models are based on the assumption that $V_1, \ldots, V_m$ are independently distributed and each follows an extreme maxima value distribution (Hoffman and Duncan 1988). An important property of such models is Independence from Irrelevant Alternatives (IIA); that is, the ratio of the choice probabilities for any two alternatives for a particular observation is not influenced systematically by any other alternatives. IIA can be tested by fitting a model that contains all the cross-alternative effects and examining the significance of these effects. The cross-alternative effects pick up a variety of IIA violations and other sources of error in the model. (See pages 452, 459, 468, and 679 for other discussions of IIA.)

In the example of travel demand, there may be separate effects for the three travel modes and travel times. In addition, there may be cross-alternative effects for travel times. Not all the effects are estimable, only two of the three intercepts and three of the six cross-alternative effects can be estimated.

The following SAS statements create the design variables for all the cross-alternative effects and display the first nine observations:

```
* Number of alternatives in each choice set;
%let m = 3;

data choice3;
   drop i j k autotime plantime trantime;

   * Values of the variable CHOSEN;
   array allmodes[&m] $
      _temporary_ ('Auto' 'Plane' 'Transit');

   * Travel times for the alternatives;
   array times[&m] autotime plantime trantime;

   * New variables that will contain the design:;
   array inters[&m]
      Auto      /*intercept for auto              */
      Plane     /*intercept for plane             */
      Transit;  /*intercept for transit           */

   array cross[%eval(&m * &m)]
      TimeAuto  /*time of auto alternative        */
      PlanAuto  /*cross-effect of plane on auto   */
      TranAuto  /*cross-effect of transit on auto */
      AutoPlan  /*cross-effect of auto on plane   */
      TimePlan  /*time of plane alternative       */
      TranPlan  /*cross-effect of transit on plane*/
      AutoTran  /*cross-effect of auto on transit */
      PlanTran  /*cross-effect of plane on transit*/
      TimeTran; /*time of transit alternative     */
   set travel;

   subject = _n_;

   * Create &m observations for each choice set;
   do i = 1 to &m;
      Mode = allmodes[i];  /* this alternative    */
      Travtime = times[i]; /* travel time         */
      Choice = 2 - (chosen eq mode);/* 1 - chosen */
      do j = 1 to &m;
         inters[j] = (i eq j);  /* mode indicator */
         do k = 1 to &m;
            * (j=k) - time, otherwise, cross-effect;
            cross[&m*(j-1)+k]=times[k]*inters[j];
            end;
         end;
      output;
      end;
   run;
```

```
proc print data=choice3(obs=9) label noobs;
   var subject mode travtime choice auto plane transit
       timeauto timeplan timetran autoplan autotran planauto
       plantran tranauto tranplan;
   run;
```

```
subject    Mode       Travtime    Choice    Auto    Plane    Transit

   1       Auto         10.0         2        1       0         0
   1       Plane         4.5         1        0       1         0
   1       Transit      10.5         2        0       0         1
   2       Auto          5.5         1        1       0         0
   2       Plane         4.0         2        0       1         0
   2       Transit       7.5         2        0       0         1
   3       Auto          4.5         2        1       0         0
   3       Plane         6.0         2        0       1         0
   3       Transit       5.5         1        0       0         1

Time     Time     Time     Auto     Auto     Plan     Plan     Tran     Tran
Auto     Plan     Tran     Plan     Tran     Auto     Tran     Auto     Plan

10.0      0.0      0.0      0.0      0.0      4.5      0.0     10.5      0.0
 0.0      4.5      0.0     10.0      0.0      0.0      0.0      0.0     10.5
 0.0      0.0     10.5      0.0     10.0      0.0      4.5      0.0      0.0
 5.5      0.0      0.0      0.0      0.0      4.0      0.0      7.5      0.0
 0.0      4.0      0.0      5.5      0.0      0.0      0.0      0.0      7.5
 0.0      0.0      7.5      0.0      5.5      0.0      4.0      0.0      0.0
 4.5      0.0      0.0      0.0      0.0      6.0      0.0      5.5      0.0
 0.0      6.0      0.0      4.5      0.0      0.0      0.0      0.0      5.5
 0.0      0.0      5.5      0.0      4.5      0.0      6.0      0.0      0.0
```

PROC PHREG allows you to specify `test` statements for testing linear hypotheses of the parameters. The test is a Wald test, which is based on the asymptotic normality of the parameter estimators. The following SAS statements invoke PROC PHREG to fit the so called "Mother Logit" model that includes all the cross-alternative effects. The TEST statement, with label IIA, specifies the null hypothesis that cross-alternative effects `AutoPlan`, `PlanTran`, and `TranAuto` are 0. Since only three cross-alternative effects are estimable and these are the first cross-alternative effects specified in the model, they account for all the cross-alternative effects in the model.

```
proc phreg data=choice3;
   model choice*choice(2) = auto plane transit timeauto timeplan
         timetran autoplan plantran tranauto planauto tranplan
         autotran / ties=breslow;
   IIA: test autoplan,plantran,tranauto;
   strata subject;
   title 'Mother Logit Model';
   run;
```

```
                            Mother Logit Model

                           The PHREG Procedure

                            Convergence Status

                Convergence criterion (GCONV=1E-8) satisfied.

                          Model Fit Statistics

                                  Without           With
                   Criterion     Covariates     Covariates

                   -2 LOG L          46.142         24.781
                   AIC               46.142         40.781
                   SBC               46.142         49.137

                   Testing Global Null Hypothesis: BETA=0

           Test                 Chi-Square       DF     Pr > ChiSq

           Likelihood Ratio        21.3607        8         0.0062
           Score                   15.4059        8         0.0517
           Wald                     6.2404        8         0.6203

                   Analysis of Maximum Likelihood Estimates

                    Parameter     Standard                                 Hazard
 Variable    DF      Estimate        Error    Chi-Square    Pr > ChiSq      Ratio

 Auto         1      -0.73812      3.05933        0.0582        0.8093      0.478
 Plane        1      -3.62435      3.48049        1.0844        0.2977      0.027
 Transit      0             0            .             .             .          .
 TimeAuto     1      -2.23433      1.89921        1.3840        0.2394      0.107
 TimePlan     1      -0.10112      0.68621        0.0217        0.8829      0.904
 TimeTran     1       0.09785      0.70096        0.0195        0.8890      1.103
 AutoPlan     1       0.44495      0.68616        0.4205        0.5167      1.560
 PlanTran     1      -0.53234      0.63481        0.7032        0.4017      0.587
 TranAuto     1       1.66295      1.51193        1.2097        0.2714      5.275
 PlanAuto     0             0            .             .             .          .
 TranPlan     0             0            .             .             .          .
 AutoTran     0             0            .             .             .          .
```

```
                  Linear Hypotheses Testing Results

                              Wald
                 Label    Chi-Square      DF    Pr > ChiSq

                 IIA          1.6526       3        0.6475
```

The χ^2 statistic for the Wald test is 1.6526 with 3 degrees of freedom, indicating that the cross-alternative effects are not statistically significant (p = .6475). A generally more preferable way of testing the significance of the cross-alternative effects is to compare the likelihood of the "Mother logit" model with the likelihood of the reduced model with the cross- alternative effects removed. The following SAS statements invoke PROC PHREG to fit the reduced model:

```
proc phreg data=choice3;
   model choice*choice(2) = auto plane transit timeauto
         timeplan timetran / ties=breslow;
   strata subject;
   title 'Reduced Model without Cross-Alternative Effects';
   run;
```

```
              Reduced Model without Cross-Alternative Effects

                           The PHREG Procedure


                            Convergence Status

               Convergence criterion (GCONV=1E-8) satisfied.

                          Model Fit Statistics

                                 Without           With
                 Criterion     Covariates     Covariates

                 -2 LOG L          46.142         27.153
                 AIC               46.142         37.153
                 SBC               46.142         42.376

                 Testing Global Null Hypothesis: BETA=0

         Test                 Chi-Square       DF     Pr > ChiSq

         Likelihood Ratio        18.9886        5         0.0019
         Score                   14.4603        5         0.0129
         Wald                     7.3422        5         0.1964
```

```
                       Analysis of Maximum Likelihood Estimates

                     Parameter     Standard                                 Hazard
Variable    DF        Estimate        Error   Chi-Square   Pr > ChiSq        Ratio

Auto         1         1.71578      1.80467       0.9039       0.3417        5.561
Plane        1        -3.60073      3.30555       1.1866       0.2760        0.027
Transit      0               0            .            .            .            .
TimeAuto     1        -0.79543      0.36327       4.7946       0.0285        0.451
TimePlan     1         0.12162      0.58954       0.0426       0.8366        1.129
TimeTran     1        -0.42184      0.25733       2.6873       0.1012        0.656
```

The chi-squared statistic for the likelihood ratio test of IIA is $(27.153 - 24.781) = 2.372$, which is not statistically significant ($p = .4989$) when compared to a χ^2 distribution with 3 degrees of freedom. This is consistent with the previous result of the Wald test. (See pages 452, 459, 468, and 674 for other discussions of IIA.)

Final Comments

For some discrete choice problems, the number of available alternatives is not the same for each individual. For example, in a study of consumer brand choices of laundry detergents as prices change, data are pooled from different locations, not all of which offer a brand that contains potash. The varying choice sets across individuals can easily be accommodated in PROC PHREG. For individual j who chooses from a set of m_j alternatives, consider m_j artificial times in which the chosen alternative has an event time 1 and the unchosen alternatives have a censored time of 2. The analysis is carried out in the same fashion as illustrated in the previous section.

Unlike the example of travel demand in which data for each individual are provided, choice data are often given in aggregate form, with choice frequencies indicating the repetition of each choice. One way of dealing with aggregate data is to expand the data to the individual level and carry out the analysis as if you have nonaggregate data. This approach is generally not recommended, because it defeats the purpose of having a smaller aggregate data set. PROC PHREG provides a FREQ statement that allows you to specify a variable that identifies the frequency of occurrence of each observation. However, with the specification of a FREQ variable, the artificial event time is no longer the only event time in a given stratum, but has ties of the given frequency. With proper stratification, the Breslow likelihood is proportional to the likelihood of the conditional logit model. Thus PROC PHREG can be used to obtain parameter estimates and hypothesis testing results for the choice models.

The `ties=discrete` option should not be used instead of the `ties=breslow` option. This is especially detrimental with aggregate choice data because the likelihood that PROC PHREG is maximizing may no longer be the same as the likelihood of the conditional logit model. `ties=discrete` corresponds to the discrete logistic model for genuinely discrete time scale, which is also suitable for the analysis of case-control studies when there is more than one case in a matched set (Gail, Lubin, and Rubinstein 1981). For nonaggregate choice data, all `ties=` options give the same results; however, the resources required for the computation are not the same, with `ties=breslow` being the most efficient.

Once you have a basic understanding of how PROC PHREG works, you can use it to fit a variety of models for the discrete choice data. The major involvement in such a task lies in reorganizing the data to create the observations necessary to form the correct risk sets and the appropriate design variables. There are many options in PROC PHREG that can also be useful in the analysis of discrete choice data. For example, the `offset=` option allows you to restrict the coefficient of an explanatory variable to the value of 1; the `selection=` option allows you to specify one of four methods for selecting variables into the model; the `outest=` option allows you to specify the name of the SAS data set that contains the parameter estimates, based on which you can easily compute the predicted probabilities of the alternatives.

This article deals with estimating parameters of discrete choice models. There is active research in the field of marketing research to use design of experiments to study consumer choice behavior. If you are interested in this area, see Carson et al. (1994), Kuhfeld et al. (1994), and Lazari et al. (1994).

Conjoint Analysis

Warren F. Kuhfeld

Abstract

Conjoint analysis is used to study consumers' product preferences and simulate consumer choice. This chapter describes conjoint analysis and provides examples using SAS. Topics include metric and nonmetric conjoint analysis, efficient experimental design, data collection and manipulation, holdouts, brand by price interactions, maximum utility and logit simulators, and change in market share.[*]

Introduction

Conjoint analysis is used to study the factors that influence consumers' purchasing decisions. Products possess attributes such as price, color, ingredients, guarantee, environmental impact, predicted reliability, and so on. Consumers typically do not have the option of buying the product that is best in every attribute, particularly when one of those attributes is price. Consumers are forced to make *trade-offs* as they decide which products to purchase. Consider the decision to purchase a car. Increased size generally means increased safety and comfort. The trade off is an increase in cost and environmental impact and a decrease in gas mileage and maneuverability. Conjoint analysis is used to study these trade-offs.

Conjoint analysis is a popular marketing research technique. It is used in designing new products, changing or repositioning existing products, evaluating the effects of price on purchase intent, and simulating market share. See Green and Rao (1971) and Green and Wind (1975) for early introductions to conjoint analysis, Louviere (1988) for a more recent introduction, and Green and Srinivasan (1990) for a review article.

Conjoint Measurement

Conjoint analysis grew out of the area of *conjoint measurement* in mathematical psychology. Conjoint measurement is used to investigate the joint effect of a set of independent variables on an ordinal-scale-of-measurement dependent variable. The independent variables are typically nominal and sometimes interval-scaled variables. Conjoint measurement simultaneously finds a monotonic scoring of the dependent variable and numerical values for each level of each independent variable. The goal is to

[*]Copies of this chapter (MR-2010H), the other chapters, sample code, and all of the macros are available on the Web `http://support.sas.com/resources/papers/tnote/tnote_marketresearch.html`. Specifically, sample code is here `http://support.sas.com/techsup/technote/mr2010h.sas`. For help, please contact SAS Technical Support. See page 25 for more information.

monotonically transform the ordinal values to equal the sum of their attribute level values. Hence, conjoint measurement is used to derive an interval variable from ordinal data. The conjoint measurement model is a mathematical model, not a statistical model, since it has no statistical error term.

Conjoint Analysis

Conjoint analysis is based on a main effects analysis-of-variance model. Subjects provide data about their preferences for hypothetical products defined by attribute combinations. Conjoint analysis decomposes the judgment data into components, based on qualitative attributes of the products. A numerical *part-worth utility* value is computed for each level of each attribute. Large part-worth utilities are assigned to the most preferred levels, and small part-worth utilities are assigned to the least preferred levels. The attributes with the largest part-worth utility range are considered the most important in predicting preference. Conjoint analysis is a statistical model with an error term and a loss function.

Metric conjoint analysis models the judgments directly. When all of the attributes are nominal, the metric conjoint analysis is a simple main-effects ANOVA with some specialized output. The attributes are the independent variables, the judgments comprise the dependent variable, and the part-worth utilities are the β's, the parameter estimates from the ANOVA model. The following formula shows a metric conjoint analysis model for three factors:

$$y_{ijk} = \mu + \beta_{1i} + \beta_{2j} + \beta_{3k} + \epsilon_{ijk}$$

where

$$\sum \beta_{1i} = \sum \beta_{2j} = \sum \beta_{3k} = 0$$

This model could be used, for example, to investigate preferences for cars that differ on three attributes: mileage, expected reliability, and price. The y_{ijk} term is one subject's stated preference for a car with the *ith* level of mileage, the *jth* level of expected reliability, and the *kth* level of price. The grand mean is μ, and the error is ϵ_{ijk}. The predicted utility for the *ijk* product is:

$$\hat{y}_{ijk} = \hat{\mu} + \hat{\beta}_{1i} + \hat{\beta}_{2j} + \hat{\beta}_{3k}$$

Nonmetric conjoint analysis finds a monotonic transformation of the preference judgments. The model, which follows directly from conjoint measurement, iteratively fits the ANOVA model until the transformation stabilizes. The R square increases during every iteration until convergence, when the change in R square is essentially zero. The following formula shows a nonmetric conjoint analysis model for three factors:

$$\Phi(y_{ijk}) = \mu + \beta_{1i} + \beta_{2j} + \beta_{3k} + \epsilon_{ijk}$$

where $\Phi(y_{ijk})$ designates a monotonic transformation of the variable `y`.

The R square for a nonmetric conjoint analysis model is always greater than or equal to the R square from a metric analysis of the same data. The smaller R square in metric conjoint analysis is not necessarily a disadvantage, since results should be more stable and reproducible with the metric model. Metric conjoint analysis was derived from nonmetric conjoint analysis as a special case. Today, metric conjoint analysis is probably used more often than nonmetric conjoint analysis.

In the SAS System, conjoint analysis is performed with the SAS/STAT procedure TRANSREG (transformation regression). Metric conjoint analysis models are fit using ordinary least squares, and nonmetric conjoint analysis models are fit using an alternating least squares algorithm (Young 1981; Gifi 1990). Conjoint analysis is explained more fully in the examples. The "PROC TRANSREG Specifications" section of this chapter starting on page 789 documents the PROC TRANSREG statements and options that are most relevant to conjoint analysis. The "Samples of PROC TRANSREG Usage" section starting on page 799 shows some typical conjoint analysis specifications. This chapter shows some of the SAS programming that is used for conjoint analysis. Alternatively, there is a marketing research GUI that performs conjoint analysis available from the main display manager PMENU by selecting: `Solutions` → `Analysis` → `Market Research`.

Choice-Based Conjoint

The meaning of the word "conjoint" has broadened over the years from conjoint measurement to conjoint analysis (which at first always meant what we now call nonmetric conjoint analysis) and later to metric conjoint analysis. Metric and nonmetric conjoint analysis are based on a linear ANOVA model. In contrast, a different technique, discrete choice, is based on the nonlinear multinomial logit model. Discrete choice is sometimes referred to as "choice-based conjoint." This technique is not discussed in this chapter, however it is discussed in detail starting on page 285.

Experimental Design

Experimental design is a fundamental component of conjoint analysis. A conjoint study uses experimental design to create a list of products that vary on an assortment of attributes such as brand, price, size, and so on, and subjects rate or rank the products. There are many examples of making conjoint designs in this chapter. Before you read them, be sure to read the design chapters beginning on pages 53 and 243.

The Output Delivery System

The Output Delivery System (ODS) can be used to customize the output of SAS procedures including PROC TRANSREG, the procedure we use for conjoint analysis. PROC TRANSREG can produce a great deal of information for conjoint analysis, more than we often wish to see. We use ODS primarily to exclude certain portions of the default conjoint output in which we are usually not interested. This creates a better, more parsimonious display for typical analyses. However, when we need it, we can revert back to getting the full array of information. See page 287 for other examples of customizing output using ODS. You can run the following step once to customize PROC TRANSREG conjoint analysis output:

```
proc template;
   edit Stat.Transreg.ParentUtilities;
      column Label Utility StdErr tValue Probt Importance Variable;
      header title;
      define title; text 'Part-Worth Utilities'; space=1; end;
      define Variable; print=off; end;
      end;
   run;
```

Running this step edits the templates for the main conjoint analysis results table and stores a copy in `sasuser`. These changes remain in effect until you delete them. These changes move the variable label to the first column, turn off displaying the variable names, and set the table header to "Part-Worth Utilities". These changes assume that each effect in the model has a variable label associated with it, so there is no need to display variable names. This is usually be the case. To return to the default output, run the following step:

```
* Delete edited template, restore original template;
proc template;
   delete Stat.Transreg.ParentUtilities;
   run;
```

By default, PROC TRANSREG displays an ANOVA table for metric conjoint analysis and both univariate and multivariate ANOVA tables for nonmetric conjoint analysis. With nonmetric conjoint analysis, PROC TRANSREG sometimes displays liberal and conservative ANOVA tables. All of the possible ANOVA tables, along with some header notes, can be suppressed by specifying the following statement before running PROC TRANSREG:

```
ods exclude notes anova liberalanova conservanova
            mvanova liberalmvanova conservmvanova;
```

For metric conjoint analysis, this statement can be abbreviated as follows:

```
ods exclude notes mvanova anova;
```

The rest of this section gives more details about what the PROC TEMPLATE step does and why. The rest of this section can be helpful if you wish to further customize the output from TRANSREG or some other procedure. Impatient readers may skip ahead to the candy example on page 687.

We are most interested in the part-worth utilities table in conjoint analysis, which contains the part-worth utilities, their standard errors, and the importance of each attribute. We can first use PROC TEMPLATE to identify the template for the utilities table and then edit the template. First, let's have PROC TEMPLATE display the templates for PROC TRANSREG. The `source stat.transreg` statement in the following step specifies that we want to see PROC TEMPLATE source code for the STAT product and the TRANSREG procedure:

```
proc template;
   source stat.transreg;
   run;
```

If we search the results for "Utilities", we find the template for the part-worth utilities table is called `Stat.Transreg.ParentUtilities`. The template is as follows:

```
define table Stat.Transreg.ParentUtilities;
   notes "Parent Utilities Table for Proc Transreg";
   dynamic FootMessages TitleText;
   column Label Utility StdErr tValue Probt Importance Variable;
   header Title;
   footer Foot;

   define Title;
      text TitleText;
      space = 1;
      spill_margin;
      first_panel;
   end;

   define Label;
      parent = Stat.Transreg.Label;
      style = RowHeader;
   end;

   define Utility;
      header = "Utility";
      format_width = 7;
      parent = Stat.Transreg.Coefficient;
   end;

   define StdErr;
      parent = Stat.Transreg.StdErr;
   end;

   define tValue;
      parent = Stat.Transreg.tValue;
      print = OFF;
   end;

   define Probt;
      parent = Stat.Transreg.Probt;
      print = OFF;
   end;

   define Importance;
      header = %nrstr(";Importance;%(%% Utility;Range%)");
      translate _val_=._ into " ";
      format = 7.3;
   end;

   define Variable;
      parent = Stat.Transreg.Variable;
   end;

   define Foot;
      text FootMessages;
      just = l;
      maximize;
   end;
```

```
      control = control;
      required_space = 20;
   end;
```

Recall that we ran the following step to customize the output:

```
   proc template;
      edit Stat.Transreg.ParentUtilities;
         column Label Utility StdErr tValue Probt Importance Variable;
         header title;
         define title; text 'Part-Worth Utilities'; space=1; end;
         define Variable; print=off; end;
         end;
      run;
```

We specify the `edit Stat.Transreg.ParentUtilities` statement to name the table that we wish to change. The `column` statement is copied from the PROC TEMPLATE source listing, and it names all of the columns in the table. Some, like `tValue` and `Probt` do not display by default. We can suppress the `Variable` column by using the `print=off` option. We redefine the table header to read "Part-Worth Utilities". The names in the `column` and `header` statements must match the names in the original template.

Chocolate Candy Example

This example illustrates conjoint analysis with rating scale data and a single subject. The subject was asked to rate his preference for eight chocolate candies. The covering was either dark or milk chocolate, the center was either chewy or soft, and the candy did or did not contain nuts. The candies were rated on a 1 to 9 scale where 1 means low preference and 9 means high preference. Conjoint analysis is used to determine the importance of each attribute and the part-worth utility for each level of each attribute.

Metric Conjoint Analysis

After data collection, the attributes and the rating data are entered into a SAS data set, for example, as follows:

```
title 'Preference for Chocolate Candies';

data choc;
   input Chocolate $ Center $ Nuts $& Rating;
   datalines;
Dark  Chewy  Nuts       7
Dark  Chewy  No Nuts    6
Dark  Soft   Nuts       6
Dark  Soft   No Nuts    4
Milk  Chewy  Nuts       9
Milk  Chewy  No Nuts    8
Milk  Soft   Nuts       9
Milk  Soft   No Nuts    7
;
```

Note that the “`&`” specification in the `input` statement is used to read character data with embedded blanks.

PROC TRANSREG is used to perform a metric conjoint analysis, for example, as follows:

```
ods exclude notes mvanova anova;
proc transreg utilities separators=', ' short;
   title2 'Metric Conjoint Analysis';
   model identity(rating) = class(chocolate center nuts / zero=sum);
   run;
```

The displayed output from the metric conjoint analysis is requested by specifying the `utilities` option in the `proc` statement. The value specified in the `separators=` option, in this case a comma followed by a blank, is used in constructing the labels for the part-worth utilities in the displayed output. With these options, the labels consist of the `class` variable name, a comma, a blank and the values of the `class` variables. We specify the `short` option to suppress the iteration history. PROC TRANSREG still displays a convergence summary table so we will know if there are any convergence problems. Since this is a metric conjoint analysis, there should be only one iteration and there should not be any problems. We specify `ods exclude notes mvanova anova` to exclude ANOVA information (which we usually want to ignore) and provide more parsimonious output. The analysis variables, the transformation of each variable, and transformation specific options are specified in the `model` statement.

The `model` statement provides for general transformation regression models, so it has a markedly different syntax from other SAS/STAT procedure `model` statements. Variable lists are specified in parentheses after a transformation name. The specification `identity(rating)` requests an `identity` transformation of the dependent variable `Rating`. A transformation name must be specified for all variable lists, even for the dependent variable in metric conjoint analysis, when no transformation is desired. The `identity` transformation of `Rating` does not change the original scoring. An equal sign follows the dependent variable specification, then the attribute variables are specified along with their transformation. The following specification designates the attributes as `class` variables with the restriction that the part-worth utilities sum to zero within each attribute:

```
class(chocolate center nuts / zero=sum)
```

A slash must be specified to separate the variables from the transformation option `zero=sum`. The `class` specification creates a main-effects design matrix from the specified variables. This example does not produce any data sets; later examples show how to store results in output SAS data sets.

The results are as follows:

```
                       Preference for Chocolate Candies
                           Metric Conjoint Analysis

                            The TRANSREG Procedure

                    Dependent Variable Identity(Rating)


                           Class Level Information

                    Class        Levels    Values

                    Chocolate         2    Dark  Milk

                    Center            2    Chewy  Soft

                    Nuts              2    No Nuts  Nuts

                 Number of Observations Read             8
                 Number of Observations Used             8

         The TRANSREG Procedure Hypothesis Tests for Identity(Rating)

               Root MSE           0.50000    R-Square    0.9500
               Dependent Mean     7.00000    Adj R-Sq    0.9125
               Coeff Var          7.14286
```

```
                          Part-Worth Utilities

                                                        Importance
                                           Standard     (% Utility
         Label               Utility          Error         Range)

         Intercept            7.0000        0.17678

         Chocolate, Dark     -1.2500        0.17678         50.000
         Chocolate, Milk      1.2500        0.17678

         Center, Chewy        0.5000        0.17678         20.000
         Center, Soft        -0.5000        0.17678

         Nuts, No Nuts       -0.7500        0.17678         30.000
         Nuts, Nuts           0.7500        0.17678
```

Recall that we used an `ods exclude` statement and we used PROC TEMPLATE on page 683 to customize the output from PROC TRANSREG.

We see `Algorithm converged` in the output indicating no problems with the iterations. We also see R square = 0.95. The last table displays the part-worth utilities. The part-worth utilities show the most and least preferred levels of the attributes. Levels with positive utility are preferred over those with negative utility. Milk chocolate (part-worth utility = 1.25) was preferred over dark (−1.25), chewy center (0.5) over soft (−0.5), and nuts (0.75) over no nuts (−0.75).

Conjoint analysis provides an approximate decomposition of the original ratings. The predicted utility for a candy is the sum of the intercept and the part-worth utilities. The conjoint analysis model for the preference for chocolate type i, center j, and nut content k is

$$y_{ijk} = \mu + \beta_{1i} + \beta_{2j} + \beta_{3k} + \epsilon_{ijk}$$

for $i = 1,2$; $j = 1,2$; $k = 1,2$; where

$$\beta_{11} + \beta_{12} = \beta_{21} + \beta_{22} = \beta_{31} + \beta_{32} = 0$$

The part-worth utilities for the attribute levels are the parameter estimates $\hat{\beta}_{11}$, $\hat{\beta}_{12}$, $\hat{\beta}_{21}$, $\hat{\beta}_{22}$, $\hat{\beta}_{31}$, and $\hat{\beta}_{32}$ from this main-effects ANOVA model. The estimate of the intercept is $\hat{\mu}$, and the error term is ϵ_{ijk}.

The predicted utility for the ijk combination is

$$\hat{y}_{ijk} = \hat{\mu} + \hat{\beta}_{1i} + \hat{\beta}_{2j} + \hat{\beta}_{3k}$$

For the most preferred milk/chewy/nuts combination, the predicted utility and actual preference values are

$$7.0 + 1.25 + 0.5 + 0.75 = 9.5 = \hat{y} \approx y = 9.0$$

For the least preferred dark/soft/no nuts combination, the predicted utility and actual preference values are

$$7.0 + -1.25 + -0.5 + -0.75 = 4.5 = \hat{y} \approx y = 4.0$$

The predicted utilities are regression predicted values; the squared correlation between the predicted utilities for each combination and the actual preference ratings is the R square.

The *importance* value is computed from the part-worth utility range for each factor (attribute). Each range is divided by the sum of all ranges and multiplied by 100. The factors with the largest part-worth utility ranges are the most important in determining preference. Note that when the attributes have a varying number of levels, attributes with the most levels sometimes have inflated importances (Wittink, Krishnamurthi, and Reibstein; 1989).

The importance values show that type of chocolate, with an importance of 50%, was the most important attribute in determining preference.

$$\frac{100 \times (1.25 - -1.25)}{(1.25 - -1.25) + (0.50 - -0.50) + (0.75 - -0.75)} = 50\%$$

The second most important attribute was whether the candy contained nuts, with an importance of 30%.

$$\frac{100 \times (0.75 - -0.75)}{(1.25 - -1.25) + (0.50 - -0.50) + (0.75 - -0.75)} = 30\%$$

Type of center was least important at 20%.

$$\frac{100 \times (0.50 - -0.50)}{(1.25 - -1.25) + (0.50 - -0.50) + (0.75 - -0.75)} = 20\%$$

Nonmetric Conjoint Analysis

In the next part of this example, PROC TRANSREG is used to perform a nonmetric conjoint analysis of the candy data set. The difference between requesting a nonmetric and metric conjoint analysis is the dependent variable transformation; a `monotone` transformation of `Rating` variable is requested instead of an `identity` transformation. Also, we did not specify the `short` option this time so that we could see the iteration history table. The `output` statement is used to put the transformed rating into the `out=` output data set. The following step performs the analysis:

```
ods graphics on;

ods exclude notes anova liberalanova conservanova
            mvanova liberalmvanova conservmvanova;
proc transreg utilities separators=', ' plots=transformations;
   title2 'Nonmetric Conjoint Analysis';
   model monotone(rating) = class(chocolate center nuts / zero=sum);
   output;
   run;
```

Nonmetric conjoint analysis iteratively derives the monotonic transformation of the ratings. Recall that we used an `ods exclude` statement and we used PROC TEMPLATE on page 683 to customize the output from PROC TRANSREG. The results are as follows:

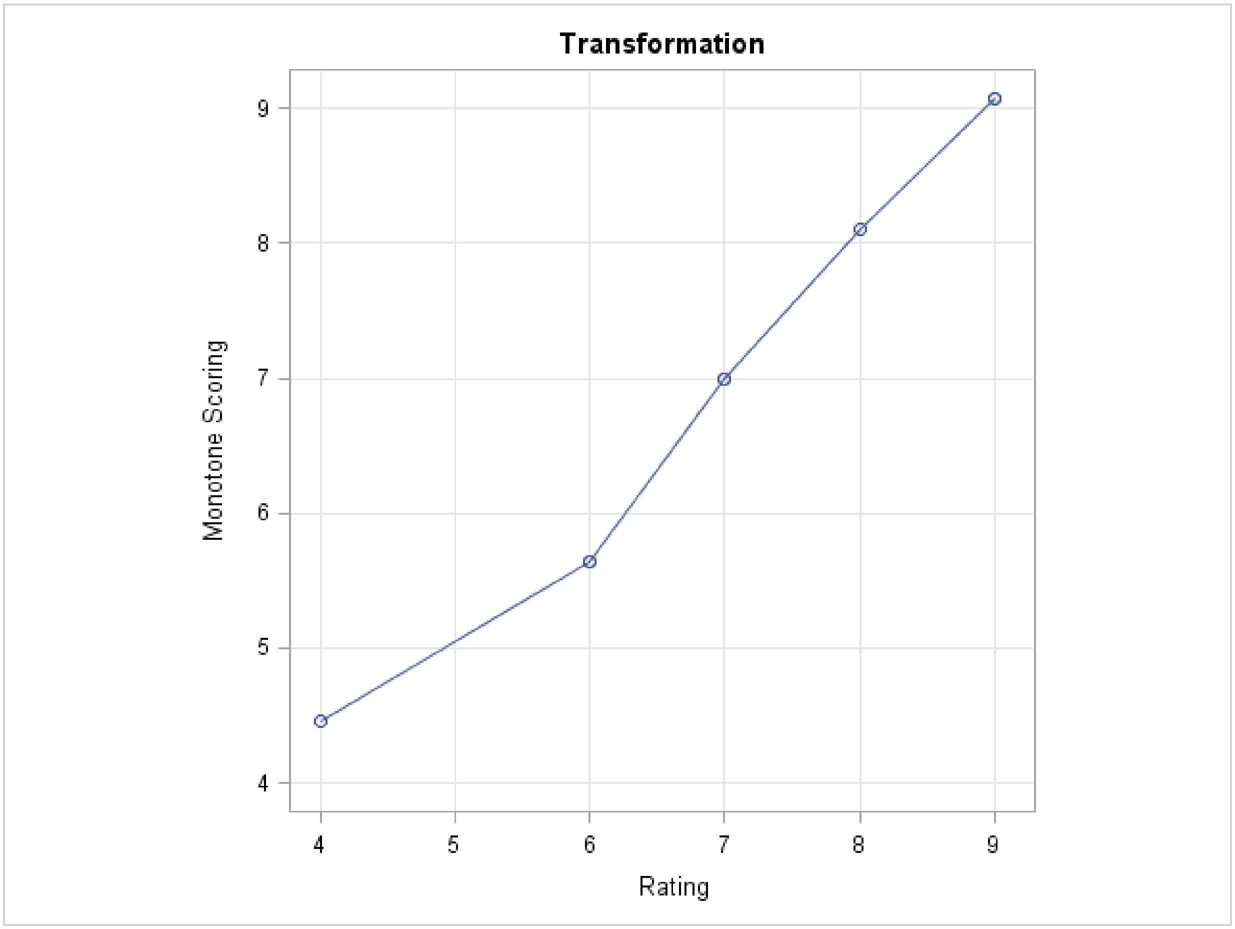
```
                       Preference for Chocolate Candies
                         Nonmetric Conjoint Analysis

                            The TRANSREG Procedure

                     Dependent Variable Monotone(Rating)


                           Class Level Information

                     Class        Levels    Values

                     Chocolate         2    Dark  Milk

                     Center            2    Chewy  Soft

                     Nuts              2    No Nuts  Nuts

                  Number of Observations Read             8
                  Number of Observations Used             8
```

```
   TRANSREG Univariate Algorithm Iteration History for Monotone(Rating)

Iteration    Average    Maximum                Criterion
   Number     Change     Change    R-Square       Change    Note
-------------------------------------------------------------------------
        1    0.08995    0.23179     0.95000
        2    0.01263    0.03113     0.96939      0.01939
        3    0.00345    0.00955     0.96981      0.00042
        4    0.00123    0.00423     0.96984      0.00003
        5    0.00050    0.00182     0.96985      0.00000
        6    0.00021    0.00078     0.96985      0.00000
        7    0.00009    0.00033     0.96985      0.00000
        8    0.00004    0.00014     0.96985      0.00000
        9    0.00002    0.00006     0.96985      0.00000
       10    0.00001    0.00003     0.96985      0.00000    Converged

Algorithm converged.

                     Preference for Chocolate Candies
                       Nonmetric Conjoint Analysis

                          The TRANSREG Procedure

       The TRANSREG Procedure Hypothesis Tests for Monotone(Rating)

             Root MSE            0.38829    R-Square    0.9698
             Dependent Mean      7.00000    Adj R-Sq    0.9472
             Coeff Var           5.54699
```

```
                         Part-Worth Utilities

                                                     Importance
                                         Standard    (% Utility
Label                  Utility            Error          Range)

Intercept               7.0000          0.13728

Chocolate, Dark        -1.3143          0.13728          53.209
Chocolate, Milk         1.3143          0.13728

Center, Chewy           0.4564          0.13728          18.479
Center, Soft           -0.4564          0.13728

Nuts, No Nuts          -0.6993          0.13728          28.312
Nuts, Nuts              0.6993          0.13728

The standard errors are not adjusted for the fact
that the dependent variable was transformed and so
are generally liberal (too small).
```

The R square increases from 0.95 for the metric case to 0.96985 for the nonmetric case. The importances and part-worth utilities are slightly different from the metric analysis, but the overall pattern of results is the same.

The transformation of the ratings is displayed with ODS Graphics as follows:



In this case, the transformation is nearly linear. In practice, the R square may increase much more than it did in this example, and the transformation may be markedly nonlinear.

Frozen Diet Entrées Example (Basic)

This example uses PROC TRANSREG to perform a conjoint analysis to study preferences for frozen diet entrées. The entrées have four attributes: three with three levels and one with two levels. The attributes are shown in the following table:

Factor	Levels		
Main Ingredient	Chicken	Beef	Turkey
Fat Claim Per Serving	8 Grams	5 Grams	2 Grams
Price	$2.59	$2.29	$1.99
Calories	350	250	

Choosing the Number of Stimuli

Ideally, for this design, we would like the number of *runs* in the experimental design to be divisible by 2 (because of the two-level factor), 3 (because of the three-level factors), $2 \times 3 = 6$ (to have equal numbers of products in each two-level and three-level factor combinations), and $3 \times 3 = 9$ (to have equal numbers of products in each pair of three-level factor combinations). If we fit a main-effects model, we need at least $1 + 3 \times (3 - 1) + (2 - 1) = 8$ runs. We can avoid doing this math ourselves and instead use the %MktRuns autocall macro to help us choose the number of products. See page 803 for macro documentation and information about installing and using SAS autocall macros. To use this macro, you specify the number of levels for each of the factors. For this example, specify three 3's and one 2. The following step invokes the macro:

```
title 'Frozen Diet Entrees';

%mktruns(3 3 3 2)
```

The results are as follows:

```
                          Frozen Diet Entrees

                            Design Summary

                         Number of
                         Levels       Frequency

                              2           1
                              3           3

                          Frozen Diet Entrees

             Saturated      = 8
             Full Factorial = 54
```

```
            Some Reasonable                        Cannot Be
               Design Sizes        Violations    Divided By

                         18 *              0
                         36 *              0
                         12                3     9
                         24                3     9
                         30                3     9
                          9                4     2 6
                         27                4     2 6
                         15                7     2 6 9
                         21                7     2 6 9
                         33                7     2 6 9
                          8 S              9     3 6 9

      * - 100% Efficient design can be made with the MktEx macro.
      S - Saturated Design - The smallest design that can be made.
          Note that the saturated design is not one of the
          recommended designs for this problem.  It is shown
          to provide some context for the recommended sizes.

                          Frozen Diet Entrees

           n    Design                          Reference

          18    2 **  1  3 **  7                Orthogonal Array
          36    2 ** 16  3 **  4                Orthogonal Array
          36    2 ** 11  3 ** 12                Orthogonal Array
          36    2 ** 10  3 **  8  6 **  1       Orthogonal Array
          36    2 **  9  3 **  4  6 **  2       Orthogonal Array
          36    2 **  4  3 ** 13                Orthogonal Array
          36    2 **  3  3 **  9  6 **  1       Orthogonal Array
          36    2 **  2  3 ** 12  6 **  1       Orthogonal Array
          36    2 **  2  3 **  5  6 **  2       Orthogonal Array
          36    2 **  1  3 **  8  6 **  2       Orthogonal Array
          36    2 **  1  3 **  3  6 **  3       Orthogonal Array
```

The output tells us that we need at least eight products, shown by the "Saturated = 8". The sizes 18 and 36 would be optimal. Twelve is a good size but three times it cannot be divided by $9 = 3 \times 3$. The "three times" comes from the $3(3-1)/2 = 3$ pairs of three-level factors. Similarly, the size 9 has four violations because it cannot be divided once by 2 and three times by $6 = 2 \times 3$ (once for each three-level factor and two-level factor pair). We could use a size smaller than 18 and not have equal frequencies everywhere, but 18 is a manageable number so we will use 18.

When an orthogonal and balanced design is available from the `%MktEx` macro, the `%MktRuns` macro tells us about it. For example, the macro tells us that our design, which is designated $2^1 3^3$, is available in 18 runs, and it can be constructed from a design with 1 two-level factor (`2 ** 1` or 2^1) and 7 three-level factors (`3 ** 7` or 3^7). Both the `%MktRuns` and `%MktEx` macros accept this '$n{*}{*}m$' exponential syntax as input, which means m factors each at n levels. Hence, `2 3 ** 7` or `2 ** 1 3 ** 7` or `2 3 3 3 3`

`3 3 3` are all equivalent level-list specifications for the experimental design $2^1 3^7$, which has 1 two-level factor and 7 three-level factors.

Generating the Design

We can use the `%MktEx` autocall macro to find a design. When you invoke the `%MktEx` macro for a simple problem, you only need to specify the numbers of levels and number of runs. The macro does the rest. The `%MktEx` macro can create designs in a number of ways. For this problem, it simply looks up an orthogonal design. The following step invokes the `%MktEx` macro:

```
%mktex(3 3 3 2, n=18)
```

The first argument to the `%MktEx` macro is a list of factor levels, and the second is the number of runs (`n=18`). These are all the options that are needed for a simple problem such as this one. However, throughout this book, random number seeds are explicitly specified with the `seed=` option so that you can reproduce these results.[*] The following steps create our design with the random number seed and the actual factor names specified:

```
%mktex(3 3 3 2, n=18, seed=151)

%mktlab(data=randomized, vars=Ingredient Fat Price Calories)
```

The `%MktEx` macro always creates factors named `x1`, `x2`, and so on. The `%MktLab` autocall macro is used to change the names when you want to provide actual factor names. This example has four factors, `Ingredient`, `Fat`, and `Price`, each with three levels and `Calories` with two levels.

The results are as follows:

```
                         Frozen Diet Entrees

                      Algorithm Search History

                             Current          Best
Design    Row,Col  D-Efficiency  D-Efficiency  Notes
----------------------------------------------------------
     1      Start      100.0000      100.0000  Tab
     1        End      100.0000
```

[*]By specifying a random number seed, results should be reproducible within a SAS release for a particular operating system. However, due to machine differences, some results may not be exactly reproducible on other machines. For most orthogonal and balanced designs, the results should be reproducible. When computerized searches are done, you might not get the same design as the one in the book, although you would expect the efficiency differences to be slight.

```
                         Frozen Diet Entrees

                         The OPTEX Procedure

                       Class Level Information

                      Class  Levels    -Values-

                         x1       3    1 2 3
                         x2       3    1 2 3
                         x3       3    1 2 3
                         x4       2    1 2
                         Frozen Diet Entrees

                         The OPTEX Procedure

                                                            Average
                                                           Prediction
Design                                                      Standard
Number     D-Efficiency     A-Efficiency     G-Efficiency      Error
------------------------------------------------------------------------
     1       100.0000         100.0000         100.0000          0.6667
```

We see that the macro had no trouble finding an optimal, 100% efficient experimental design. The value `Tab` in the `Notes` column of the algorithm search history tells us the macro was able to find the design in the `%MktEx` macro's large table (or catalog) of orthogonal arrays. In contrast, the other designs that `%MktEx` can make are algorithmically generated by the computer or generated in part from an orthogonal array and in part algorithmically. See pages 803, 1017, and the discrete choice examples starting on page 285 for more information about how the `%MktEx` macro works.

The `%MktEx` macro creates two output data sets with the experimental design, `Design` and `Randomized`. The `Design` data set is sorted. A number of the orthogonal arrays often have a first row consisting entirely of ones. For these reasons, you should typically use the *randomized* design. In the randomized design, the profiles are presented in a random order and the levels have been randomly reassigned. Neither of these operations affects the design efficiency, balance, or orthogonality. When there are restrictions on the design (see, for example, page 754), the profiles are sorted into a random order, but the levels are *not* randomly reassigned. The randomized design is the default input to the `%MktLab` macro.

Evaluating and Preparing the Design

We use the FORMAT procedure to create descriptive labels for the levels of the attributes. By default, the values of the factors are positive integers. For example, for `ingredient`, we create a format `if` (for Ingredient Format) that assigns the descriptive value label "Chicken" for level 1, "Beef" for level 2, and "Turkey" for level 3. A permanent SAS data set is created with the formats assigned (although, as we will see in the next example, we could have done this previously in the `%MktLab` step). The following steps format and display the design:

```
proc format;
   value if  1='Chicken'  2='Beef'     3='Turkey';
   value ff  1='8 Grams'  2='5 Grams'  3='2 Grams';
   value pf  1='$2.59'    2='$2.29'    3='$1.99';
   value cf  1='350'      2='250';
   run;

data sasuser.dietdes;
   set final;
   format ingredient if. fat ff. price pf. calories cf.;
   run;

proc print; run;
```

The design is as follows:

```
                        Frozen Diet Entrees

         Obs    Ingredient      Fat       Price    Calories

           1     Turkey       5 Grams    $1.99       350
           2     Turkey       8 Grams    $2.29       350
           3     Chicken      8 Grams    $1.99       350
           4     Turkey       2 Grams    $2.59       250
           5     Beef         8 Grams    $2.59       350
           6     Beef         2 Grams    $1.99       350
           7     Beef         5 Grams    $2.29       350
           8     Beef         5 Grams    $2.29       250
           9     Chicken      2 Grams    $2.29       350
          10     Beef         8 Grams    $2.59       250
          11     Turkey       8 Grams    $2.29       250
          12     Chicken      5 Grams    $2.59       350
          13     Chicken      5 Grams    $2.59       250
          14     Chicken      2 Grams    $2.29       250
          15     Turkey       5 Grams    $1.99       250
          16     Turkey       2 Grams    $2.59       350
          17     Beef         2 Grams    $1.99       250
          18     Chicken      8 Grams    $1.99       250
```

Even when you know the design is 100% *D*-efficient, orthogonal, and balanced, it is good to run basic checks on your designs. You can use the `%MktEval` autocall macro as follows to display information about the design:

```
%mkteval(data=sasuser.dietdes)
```

The macro first displays a matrix of canonical correlations between the factors. We hope to see an identity matrix (a matrix of ones on the diagonal and zeros everywhere else), which would mean that all of the factors are uncorrelated. Next, the macro displays all one-way frequencies for all attributes, all two-way frequencies, and all n-way frequencies (in this case four-way frequencies). We hope to see equal or at least nearly equal one-way and two-way frequencies, and we want to see that each combination occurs only once. The results are as follows:

```
                         Frozen Diet Entrees
                Canonical Correlations Between the Factors
           There are 0 Canonical Correlations Greater Than 0.316

                    Ingredient    Fat     Price    Calories

       Ingredient        1         0        0          0
       Fat               0         1        0          0
       Price             0         0        1          0
       Calories          0         0        0          1

                         Frozen Diet Entrees
                        Summary of Frequencies
           There are 0 Canonical Correlations Greater Than 0.316

                               Frequencies

       Ingredient              6 6 6
       Fat                     6 6 6
       Price                   6 6 6
       Calories                9 9
       Ingredient Fat          2 2 2 2 2 2 2 2 2
       Ingredient Price        2 2 2 2 2 2 2 2 2
       Ingredient Calories     3 3 3 3 3 3
       Fat Price               2 2 2 2 2 2 2 2 2
       Fat Calories            3 3 3 3 3 3
       Price Calories          3 3 3 3 3 3
       N-Way                   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

A *canonical correlation* is the maximum correlation between linear combinations of the coded factors (see page 101). All zeros off the diagonal show that this design is orthogonal for main effects. If any off-diagonal canonical correlations had been greater than 0.316 ($r^2 > 0.1$), the macro would have listed them in a separate table. The last title line tells you that none of them were this large. For nonorthogonal designs and designs with interactions, the canonical-correlation matrix is not a substitute for looking at the variance matrix (with `examine=v`) in the `%MktEx` macro. The `%MktEx` macro just provides a quick and more-compact picture of the correlations between the factors. The variance matrix is sensitive to the actual model specified and the coding. The canonical-correlation matrix just tells you if there is some correlation between the main effects. In this case, there are no correlations.

The equal one-way frequencies show you that this design is balanced. The equal two-way frequencies show you that this design is orthogonal. Equal one-way and two-way frequencies together show you that this design is 100% D-efficient. The n-way frequencies, all equal to one, show you that there are no duplicate profiles. This is a perfect design for a main effects model. However, there are other 100% efficient designs for this problem with duplicate observations. In the last part of the output, the n-Way frequencies may contain some 2's for those designs. You can specify `options=nodups` in the `%MktEx` macro to ensure that there are no duplicate profiles.

The `%MktEval` macro produces a very compact summary of the design, hence some information, for example, the levels to which the frequencies correspond, is not shown. You can use the `print=freqs` option in the `%MktEval` macro to get a less compact and more detailed display.

An alternative way to check for duplicate profiles is the `%MktDups` macro. You must specify that this is a linear model design (as opposed to a choice design) and name the data set with the design to evaluate. By default, all numeric variables are used. The following step invokes the macro:

```
%mktdups(linear, data=sasuser.dietdes)
```

The results are as follows:

```
Design:          Linear
Factors:         _numeric_
                 Calories Fat Ingredient Price
Duplicate Runs:  0
```

This output shows that there are no duplicate profiles, but we already knew that from the `%MktEval` results.

Printing the Stimuli and Data Collection

Next, we generate the stimuli using the following DATA step:

```
title;
data _null_;
   file print;
   set sasuser.dietdes;
   put ///
       +3 ingredient 'Entree' @50 '(' _n_ +(-1) ')' /
       +3 'With ' fat 'of Fat and ' calories  'Calories' /
       +3 'Now for Only ' Price +(-1) '.'///;
   if mod(_n_, 6) = 0 then put _page_;
   run;
```

The `data _null_` step uses the `file` statement to set the print destination to the printed output destination. The design data set is read with the `set` statement. A `put` statement prints the attributes along with some constant text and the combination number. The `put` statement option `+3` skips 3 spaces, `@50` starts printing in column 50, `+(-1)` skips one space *backwards* getting rid of the blank that would by default appear after the stimulus number, and `/` skips to a new line. Text enclosed in quotes is literally copied to the output. For our attribute variables, the formatted values are printed. The variable `_n_` is the number of the current pass through the DATA step, which in this case is the stimulus number. The `if` statement causes six descriptions to be printed on a page. The results are as follows:

```
Turkey Entree                                  (1)
With 5 Grams of Fat and 350 Calories
Now for Only $1.99.

Turkey Entree                                  (2)
With 8 Grams of Fat and 350 Calories
Now for Only $2.29.

Chicken Entree                                 (3)
With 8 Grams of Fat and 350 Calories
Now for Only $1.99.

Turkey Entree                                  (4)
With 2 Grams of Fat and 250 Calories
Now for Only $2.59.

Beef Entree                                    (5)
With 8 Grams of Fat and 350 Calories
Now for Only $2.59.

Beef Entree                                    (6)
With 2 Grams of Fat and 350 Calories
Now for Only $1.99.

Beef Entree                                    (7)
With 5 Grams of Fat and 350 Calories
Now for Only $2.29.

Beef Entree                                    (8)
With 5 Grams of Fat and 250 Calories
Now for Only $2.29.

Chicken Entree                                 (9)
With 2 Grams of Fat and 350 Calories
Now for Only $2.29.

Beef Entree                                    (10)
With 8 Grams of Fat and 250 Calories
Now for Only $2.59.

Turkey Entree                                  (11)
With 8 Grams of Fat and 250 Calories
Now for Only $2.29.

Chicken Entree                                 (12)
With 5 Grams of Fat and 350 Calories
Now for Only $2.59.
```

```
Chicken Entree                                  (13)
With 5 Grams of Fat and 250 Calories
Now for Only $2.59.

Chicken Entree                                  (14)
With 2 Grams of Fat and 250 Calories
Now for Only $2.29.

Turkey Entree                                   (15)
With 5 Grams of Fat and 250 Calories
Now for Only $1.99.

Turkey Entree                                   (16)
With 2 Grams of Fat and 350 Calories
Now for Only $2.59.

Beef Entree                                     (17)
With 2 Grams of Fat and 250 Calories
Now for Only $1.99.

Chicken Entree                                  (18)
With 8 Grams of Fat and 250 Calories
Now for Only $1.99.
```

Next, we print the stimuli, produce the cards, and ask a subject to sort the cards from most preferred to least preferred. The combination numbers (most preferred to least preferred) are entered as data. For example, this subject's most preferred combination is 17, which is the "Beef Entree, With 2 Grams of Fat and 250 Calories, Now for Only $1.99", and her least preferred combination is 18, "Chicken Entree, With 8 Grams of Fat and 250 Calories, Now for Only $1.99".

Data Processing

The data are transposed, going from one observation and 18 variables to 18 observations and one variable named `Combo`. The next DATA step creates the variable `Rank`: 1 for the first and most preferred combination, ..., and 18 for the last and least preferred combination. The following steps sort the data by combination number and merge them with the design:

```
title 'Frozen Diet Entrees';

data results;
   input combo1-combo18;
   datalines;
17 6 8 7 10 5 4 16 15 1 11 2 9 14 12 13 3 18
;

proc transpose out=results(rename=(col1=combo)); run;

data results; set results; Rank = _n_; drop _name_; run;
```

```
proc sort; by combo; run;

data results(drop=combo);
   merge sasuser.dietdes results;
   run;

proc print; run;
```

The results are as follows:

```
                         Frozen Diet Entrees

 Obs    Ingredient      Fat       Price    Calories    Rank

   1     Turkey       5 Grams    $1.99       350        10
   2     Turkey       8 Grams    $2.29       350        12
   3     Chicken      8 Grams    $1.99       350        17
   4     Turkey       2 Grams    $2.59       250         7
   5     Beef         8 Grams    $2.59       350         6
   6     Beef         2 Grams    $1.99       350         2
   7     Beef         5 Grams    $2.29       350         4
   8     Beef         5 Grams    $2.29       250         3
   9     Chicken      2 Grams    $2.29       350        13
  10     Beef         8 Grams    $2.59       250         5
  11     Turkey       8 Grams    $2.29       250        11
  12     Chicken      5 Grams    $2.59       350        15
  13     Chicken      5 Grams    $2.59       250        16
  14     Chicken      2 Grams    $2.29       250        14
  15     Turkey       5 Grams    $1.99       250         9
  16     Turkey       2 Grams    $2.59       350         8
  17     Beef         2 Grams    $1.99       250         1
  18     Chicken      8 Grams    $1.99       250        18
```

Recall that the seventeenth combination was most preferred, and it has a rank of 1. The eighteenth combination was least preferred and it has a rank of 18.

Nonmetric Conjoint Analysis

You can use PROC TRANSREG to perform the nonmetric conjoint analysis of the ranks as follows:

```
ods exclude notes anova liberalanova conservanova
            mvanova liberalmvanova conservmvanova;
proc transreg utilities order=formatted separators=', ';
   model monotone(rank / reflect) =
         class(Ingredient Fat Price Calories / zero=sum);
   output out=utils p ireplace;
   run;
```

The utilities option displays the part-worth utilities and importance table. The order=formatted option sorts the levels of the attributes by the formatted values. By default, levels are sorted by their internal unformatted values (in this case the integers 1, 2, 3). The model statement names the variable Rank as the dependent variable and specifies a monotone transformation for the nonmetric conjoint analysis. The reflect transformation option is specified with rank data. With rank data, small values mean high preference and large values mean low preference. The reflect transformation option reflects the ranks around their mean (–(rank – mean rank) + mean rank) so that in the results, large part-worth utilities mean high preference. With ranks ranging from 1 to 18, reflect transforms 1 to 18, 2 to 17, ..., r to $(19 - r)$, ..., and 18 to 1. (Note that the mean rank is the midpoint, in this case $(18 + 1)/2 = 9.5$, and $-(r - \bar{r}) + \bar{r} = 2\bar{r} - r = 2(\max(r) + \min(r))/2 - r = 19 - r$.) The class specification names the attributes and scales the part-worth utilities to sum to zero within each attribute.

The output statement creates the out= data set, which contains the original variables, transformed variables, and indicator variables. The predicted utilities for all combinations are written to this data set by the p option (for predicted values). The ireplace option specifies that the transformed independent variables replace the original independent variables, since both are the same.

The results of the conjoint analysis are as follows:

```
                          Frozen Diet Entrees

                        The TRANSREG Procedure

                 Dependent Variable Monotone(Rank)

                       Class Level Information

              Class         Levels    Values

              Ingredient         3    Beef  Chicken  Turkey

              Fat                3    2 Grams  5 Grams  8 Grams

              Price              3    $1.99  $2.29  $2.59

              Calories           2    250  350

                  Number of Observations Read            18
                  Number of Observations Used            18
```

```
    TRANSREG Univariate Algorithm Iteration History for Monotone(Rank)

Iteration    Average    Maximum                Criterion
   Number     Change     Change    R-Square       Change    Note
-------------------------------------------------------------------------
        1    0.07276    0.10014     0.99174
        2    0.00704    0.01074     0.99977      0.00802
        3    0.00468    0.00710     0.99990      0.00013
        4    0.00311    0.00470     0.99995      0.00006
        5    0.00207    0.00312     0.99998      0.00003
        6    0.00138    0.00208     0.99999      0.00001
        7    0.00092    0.00138     1.00000      0.00001
        8    0.00061    0.00092     1.00000      0.00000
        9    0.00041    0.00061     1.00000      0.00000
       10    0.00027    0.00041     1.00000      0.00000
       11    0.00018    0.00027     1.00000      0.00000
       12    0.00012    0.00018     1.00000      0.00000
       13    0.00008    0.00012     1.00000      0.00000
       14    0.00005    0.00008     1.00000      0.00000
       15    0.00004    0.00005     1.00000      0.00000
       16    0.00002    0.00004     1.00000      0.00000
       17    0.00002    0.00002     1.00000      0.00000
       18    0.00001    0.00002     1.00000      0.00000
       19    0.00001    0.00001     1.00000      0.00000    Converged

Algorithm converged.

                          Frozen Diet Entrees

                         The TRANSREG Procedure

      The TRANSREG Procedure Hypothesis Tests for Monotone(Rank)


             Root MSE         0.00007166    R-Square    1.0000
             Dependent Mean      9.50000    Adj R-Sq    1.0000
             Coeff Var        0.00075429
```

```
                        Part-Worth Utilities

                                                       Importance
                                          Standard    (% Utility
Label                         Utility        Error        Range)

Intercept                      9.5000      0.00002

Ingredient, Beef               6.0281      0.00002        74.999
Ingredient, Chicken           -6.0281      0.00002
Ingredient, Turkey            -0.0000      0.00002

Fat, 2 Grams                   2.0094      0.00002        25.000
Fat, 5 Grams                   0.0000      0.00002
Fat, 8 Grams                  -2.0094      0.00002

Price, $1.99                   0.0000      0.00002         0.000
Price, $2.29                   0.0000      0.00002
Price, $2.59                  -0.0000      0.00002

Calories, 250                  0.0001      0.00002         0.001
Calories, 350                 -0.0001      0.00002

The standard errors are not adjusted for the fact that
the dependent variable was transformed and so are
generally liberal (too small).
```

Recall that we used an `ods exclude` statement and we used PROC TEMPLATE on page 683 to customize the output from PROC TRANSREG.

We see in the conjoint output that main ingredient was the most important attribute at almost 75% and that beef was preferred over turkey, which was preferred over chicken. We also see that fat content was the second most important attribute at 25% and lower fat is preferred over higher fat. Price and calories only account for essentially none of the preference.

The following steps sort the products in the `out=` data set by their predicted utility and displays them along with their rank, transformed and reflected rank, and predicted values (predicted utility):

```
proc sort; by descending prank; run;

proc print label;
   var ingredient fat price calories rank trank prank;
   label trank = 'Reflected Rank'
         prank = 'Utilities';
   run;
```

The results are as follows:

```
                                 Frozen Diet Entrees

                                                             Reflected
    Obs    Ingredient      Fat      Price    Calories    Rank       Rank      Utilities

      1     Beef         2 Grams    $1.99      250          1     17.5375      17.5375
      2     Beef         2 Grams    $1.99      350          2     17.5373      17.5373
      3     Beef         5 Grams    $2.29      250          3     15.5282      15.5281
      4     Beef         5 Grams    $2.29      350          4     15.5279      15.5280
      5     Beef         8 Grams    $2.59      250          5     13.5188      13.5188
      6     Beef         8 Grams    $2.59      350          6     13.5186      13.5186
      7     Turkey       2 Grams    $2.59      250          7     11.5095      11.5094
      8     Turkey       2 Grams    $2.59      350          8     11.5092      11.5093
      9     Turkey       5 Grams    $1.99      250          9      9.5001       9.5001
     10     Turkey       5 Grams    $1.99      350         10      9.4999       9.4999
     11     Turkey       8 Grams    $2.29      250         11      7.4908       7.4907
     12     Turkey       8 Grams    $2.29      350         12      7.4905       7.4906
     13     Chicken      2 Grams    $2.29      250         14      5.4813       5.4814
     14     Chicken      2 Grams    $2.29      350         13      5.4813       5.4812
     15     Chicken      5 Grams    $2.59      250         16      3.4719       3.4720
     16     Chicken      5 Grams    $2.59      350         15      3.4719       3.4719
     17     Chicken      8 Grams    $1.99      250         18      1.4626       1.4627
     18     Chicken      8 Grams    $1.99      350         17      1.4626       1.4625
```

The variable `Rank` is the original rank variable; `TRank` contains the transformation of rank, in this case the reflection and monotonic transformation; and `PRank` contains the predicted utilities or predicted values. The first letter of the variable name comes from the first letter of "Transformation" and "Predicted".

It is interesting to see that the sorted combinations support the information in the utilities table. The combinations are perfectly sorted on beef, turkey, and chicken. Furthermore, within ties in the main ingredient, the products are sorted by fat content.

Frozen Diet Entrées Example (Advanced)

This example is an advanced version of the previous example. It illustrates conjoint analysis with more than one subject. It has six parts.

- The `%MktEx` macro is used to generate an experimental design.
- Holdout observations are generated.
- The descriptions of the products are printed for data collection.
- The data are collected, entered, and processed.
- The metric conjoint analysis is performed.
- Results are summarized across subjects.

Creating a Design with the %MktEx Macro

The first thing you need to do in a conjoint study is decide on the product attributes and levels. Then you create the experimental design. We use the same experimental design as we used in the previous example. The attributes and levels are shown in the table.

Factor	Levels		
Main Ingredient	Chicken	Beef	Turkey
Fat Claim Per Serving	8 Grams	5 Grams	2 Grams
Price	$2.59	$2.29	$1.99
Calories	350	250	

We create our designs in the same way as we did in the previous example, starting on page 697. Only the random number seed has changed. Like before, we use the `%MktEval` macro to check the one-way and two-way frequencies and to ensure that each combination only appears once. See page 803 for macro documentation and information about installing and using SAS autocall macros. The following steps create and evaluate the design:

```
title 'Frozen Diet Entrees';

proc format;
   value if  1='Chicken'  2='Beef'     3='Turkey';
   value ff  1='8 Grams'  2='5 Grams'  3='2 Grams';
   value pf  1='$2.59'    2='$2.29'    3='$1.99';
   value cf  1='350'      2='250';
   run;

%mktex(3 3 3 2, n=18, seed=205)

%mktlab(data=randomized, vars=Ingredient Fat Price Calories)

%mkteval(data=final)
```

The results are as follows:

```
                              Frozen Diet Entrees

                            Algorithm Search History

                                Current          Best
          Design    Row,Col  D-Efficiency  D-Efficiency  Notes
          ----------------------------------------------------------
               1      Start      100.0000      100.0000  Tab
               1        End      100.0000

                              Frozen Diet Entrees

                              The OPTEX Procedure

                            Class Level Information

                             Class  Levels    -Values-

                               x1       3    1 2 3
                               x2       3    1 2 3
                               x3       3    1 2 3
                               x4       2    1 2

                              Frozen Diet Entrees

                              The OPTEX Procedure

                                                                 Average
                                                                Prediction
   Design                                                        Standard
   Number     D-Efficiency     A-Efficiency     G-Efficiency       Error
   ------------------------------------------------------------------------
        1         100.0000         100.0000         100.0000        0.6667

                              Frozen Diet Entrees
                    Canonical Correlations Between the Factors
              There are 0 Canonical Correlations Greater Than 0.316

                             Ingredient    Fat     Price    Calories

              Ingredient        1           0        0          0
              Fat               0           1        0          0
              Price             0           0        1          0
              Calories          0           0        0          1
```

```
                    Frozen Diet Entrees
                   Summary of Frequencies
         There are 0 Canonical Correlations Greater Than 0.316

                            Frequencies

         Ingredient             6 6 6
         Fat                    6 6 6
         Price                  6 6 6
         Calories               9 9
         Ingredient Fat         2 2 2 2 2 2 2 2 2
         Ingredient Price       2 2 2 2 2 2 2 2 2
         Ingredient Calories    3 3 3 3 3 3
         Fat Price              2 2 2 2 2 2 2 2 2
         Fat Calories           3 3 3 3 3 3
         Price Calories         3 3 3 3 3 3
         N-Way                  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

This design is 100% efficient, perfectly balanced and perfectly orthogonal. The *n*-way frequencies show us that each of the 18 hypothetical products occurs exactly once, so there are no duplicate profiles.

Designing Holdouts

The next steps add *holdout* observations to the design and display the results. Holdouts are ranked by the subjects but are analyzed with zero weight to exclude them from contributing to the utility computations. The correlation between the ranks for holdouts and their predicted utilities provide an indication of the validity of the results of the study. The following steps create and evaluate the design:

```
title 'Frozen Diet Entrees';

proc format;
   value if  1='Chicken'  2='Beef'     3='Turkey';
   value ff  1='8 Grams'  2='5 Grams'  3='2 Grams';
   value pf  1='$2.59'    2='$2.29'    3='$1.99';
   value cf  1='350'      2='250';
   run;

%mktex(3 3 3 2, n=18, seed=205)

%mktex(3 3 3 2,                    /* 3 three-level and a two-level factor */
       n=22,                       /* 22 runs                              */
       init=randomized,            /* initial design                       */
       holdouts=4,                 /* add four holdouts to init design     */
       options=nodups,             /* no duplicate rows in design          */
       seed=368)                   /* random number seed                   */
```

```
proc print data=randomized; run;

%mkteval(data=randomized(where=(w=1)), factors=x:)
%mkteval(data=randomized(drop=w))

%mktlab(data=randomized, out=sasuser.dietdes,
        vars=Ingredient Fat Price Calories,
        statements=format Ingredient if. fat ff. price pf. calories cf.)

proc print; run;
```

The first `%MktEx` step recreates the formats and the design (just so you can see all of the code for a design with holdouts in one step). The next `%MktEx` step adds four holdouts to the randomized design created from the previous step. The specification `options=nodups` (no duplicates) ensures that the holdouts do not match products already in the design. The first `%MktEval` step evaluates just the original design, excluding the holdouts. The second `%MktEval` step evaluates the entire design. Both `%MktEval` steps ensure that the variable `w`, which flags the active and holdout observations, is excluded and not treated as a factor. The `%MktLab` step gives the factors informative names and assigns formats. Unlike the previous examples, this time we directly assign the formats in the `%MktLab` macro using the `statements=` option, specifying a complete `format` statement.

The last part of the output from the first `%MktEx` step, which shows that the macro found a 100% efficient design, is as follows:

```
                              Frozen Diet Entrees

                             The OPTEX Procedure

                                                                Average
                                                              Prediction
       Design                                                  Standard
       Number     D-Efficiency     A-Efficiency     G-Efficiency       Error
       ------------------------------------------------------------------------
            1         100.0000         100.0000         100.0000          0.6667
```

The following results contain some of the output from the `%MktEx` step that finds the holdouts:

```
                          Frozen Diet Entrees

                       Design Refinement History

                               Current         Best
     Design    Row,Col  D-Efficiency  D-Efficiency  Notes
     ----------------------------------------------------------
          0    Initial       98.0764                Ini

          1      Start       98.0764                Pre,Mut,Ann
          1     22   1       98.2421       98.2421  Conforms
          1        End       98.2421

          2      Start       98.2421                Pre,Mut,Ann
          2      2   1       98.2421       98.2421  Conforms
          2        End       98.2421

          3      Start       98.2421                Pre,Mut,Ann
          3      2   1       98.2421       98.2421  Conforms
          3        End       98.2421

          4      Start       98.2421                Pre,Mut,Ann
          4      2   1       98.2421       98.2421  Conforms
          4        End       98.2421

          5      Start       98.2421                Pre,Mut,Ann
          5      2   1       98.2421       98.2421  Conforms
          5        End       98.2421

NOTE: Stopping since it appears that no improvement is possible.
```

Notice that the macro immediately enters the design refinement step.

The design is as follows:

```
                          Frozen Diet Entrees

                    Obs    x1    x2    x3    x4    w

                      1     2     3     1     2    .
                      2     2     2     1     2    1
                      3     3     3     3     1    1
                      4     3     3     3     2    1
                      5     3     1     1     1    1
                      6     1     3     1     1    1
```

```
  7     1     3     1     2    1
  8     1     1     2     1    1
  9     2     2     1     1    1
 10     3     2     2     2    1
 11     2     2     3     1    .
 12     2     3     2     2    1
 13     3     1     1     2    1
 14     2     1     3     2    1
 15     3     2     1     1    .
 16     1     2     3     1    1
 17     1     1     2     2    1
 18     1     2     2     2    .
 19     2     3     2     1    1
 20     3     2     2     1    1
 21     1     2     3     2    1
 22     2     1     3     1    1
```

Observations with `w` equal to 1 comprise the original design. The observations with a missing `w` are the holdouts.

The results of the evaluation of the original design are as follows:

```
                  Frozen Diet Entrees
        Canonical Correlations Between the Factors
   There are 0 Canonical Correlations Greater Than 0.316

               x1      x2      x3      x4

         x1    1       0       0       0
         x2    0       1       0       0
         x3    0       0       1       0
         x4    0       0       0       1
```

```
                         Frozen Diet Entrees
                        Summary of Frequencies
         There are 0 Canonical Correlations Greater Than 0.316

                        Frequencies

               x1       6 6 6
               x2       6 6 6
               x3       6 6 6
               x4       9 9
               x1 x2    2 2 2 2 2 2 2 2 2
               x1 x3    2 2 2 2 2 2 2 2 2
               x1 x4    3 3 3 3 3 3
               x2 x3    2 2 2 2 2 2 2 2 2
               x2 x4    3 3 3 3 3 3
               x3 x4    3 3 3 3 3 3
               N-Way    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

The evaluation of the design with holdouts is as follows:

```
                         Frozen Diet Entrees
               Canonical Correlations Between the Factors
         There are 0 Canonical Correlations Greater Than 0.316

                        x1      x2      x3      x4

                  x1    1       0.09    0.17    0.11
                  x2    0.09    1       0.09    0.11
                  x3    0.17    0.09    1       0.11
                  x4    0.11    0.11    0.11    1
```

```
                   Frozen Diet Entrees
                  Summary of Frequencies
      There are 0 Canonical Correlations Greater Than 0.316
            * - Indicates Unequal Frequencies

                  Frequencies

 *    x1       7 8 7
 *    x2       6 9 7
 *    x3       8 7 7
      x4       11 11
 *    x1 x2    2 3 2 2 3 3 2 3 2
 *    x1 x3    2 3 2 3 2 3 3 2 2
 *    x1 x4    3 4 4 4 4 3
 *    x2 x3    2 2 2 3 3 3 3 2 2
 *    x2 x4    3 3 5 4 3 4
 *    x3 x4    4 4 3 4 4 3
      N-Way    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
               1 1 1
```

The design, displayed with descriptive factor names and formats, is as follows:

```
                   Frozen Diet Entrees

 Obs    Ingredient      Fat      Price    Calories    w

   1     Beef         2 Grams    $2.59      250       .
   2     Beef         5 Grams    $2.59      250       1
   3     Turkey       2 Grams    $1.99      350       1
   4     Turkey       2 Grams    $1.99      250       1
   5     Turkey       8 Grams    $2.59      350       1

   6     Chicken      2 Grams    $2.59      350       1
   7     Chicken      2 Grams    $2.59      250       1
   8     Chicken      8 Grams    $2.29      350       1
   9     Beef         5 Grams    $2.59      350       1
  10     Turkey       5 Grams    $2.29      250       1

  11     Beef         5 Grams    $1.99      350       .
  12     Beef         2 Grams    $2.29      250       1
  13     Turkey       8 Grams    $2.59      250       1
  14     Beef         8 Grams    $1.99      250       1
  15     Turkey       5 Grams    $2.59      350       .
```

```
                  16     Chicken      5 Grams    $1.99      350        1
                  17     Chicken      8 Grams    $2.29      250        1
                  18     Chicken      5 Grams    $2.29      250        .
                  19     Beef         2 Grams    $2.29      350        1
                  20     Turkey       5 Grams    $2.29      350        1
                  21     Chicken      5 Grams    $1.99      250        1
                  22     Beef         8 Grams    $1.99      350        1
```

Print the Stimuli

Once the design is generated, the *stimuli* (descriptions of the combinations) must be generated for data collection. They are printed using the exact same step that we used on page 701. The following step displays the stimuli:

```
title;
data _null_;
   file print;
   set sasuser.dietdes;
   put ///
       +3 ingredient 'Entree' @50 '(' _n_ +(-1) ')' /
       +3 'With ' fat 'of Fat and ' calories  'Calories' /
       +3 'Now for Only ' Price +(-1) '.'///;
   if mod(_n_, 6) = 0 then put _page_;
   run;
```

In the interest of space, only the first three stimuli are shown as follows:

```
   Beef Entree                                    (1)
   With 2 Grams of Fat and 250 Calories
   Now for Only $2.59.

   Beef Entree                                    (2)
   With 5 Grams of Fat and 250 Calories
   Now for Only $2.59.

   Turkey Entree                                  (3)
   With 2 Grams of Fat and 350 Calories
   Now for Only $1.99.
```

Data Collection, Entry, and Preprocessing

The next step in the conjoint analysis study is data collection and entry. Each subject was asked to take the 22 cards and rank them from the most preferred combination to the least preferred combination. The combination numbers are entered as data. The data follow the `datalines` statement in the next DATA step. For the first subject, 4 was most preferred, 3 was second most preferred, ..., and 5 was the least preferred combination. The following DATA step validates the data entry and converts the input to ranks:

```
   title 'Frozen Diet Entrees';

   %let m = 22; /* number of combinations */

   * Read the input data and convert to ranks;
   data ranks(drop=i k c1-c&m);
      input c1-c&m;
      array c[&m];
      array r[&m];
      do i = 1 to &m;
         k = c[i];
         if 1 le k le &m then do;
            if r[k] ne . then
               put 'ERROR: For subject ' _n_ +(-1) ', combination ' k
                   'is given more than once.';
            r[k] = i; /* Convert to ranks. */
            end;
         else put 'ERROR: For subject ' _n_ +(-1) ', combination ' k
                   'is invalid.';
         end;

      do i = 1 to &m;
         if r[i] = . then
            put 'ERROR: For subject ' _n_ +(-1) ', combination ' i
                'is not given.';
         end;
      name = 'Subj' || put(_n_, z2.);
      datalines;
   4 3 7 21 12 10 6 19 1 16 18 11 20 14 17 15 2 22 9 8 13 5
   4 12 3 1 19 7 10 6 11 21 16 2 18 20 15 9 14 22 13 17 5 8
   4 3 7 12 19 21 1 6 10 18 16 11 20 15 2 14 9 17 22 8 13 5
   4 12 1 10 21 14 18 3 7 2 17 13 19 11 22 20 16 15 6 9 5 8
   4 21 14 11 16 3 12 22 19 18 10 17 8 20 7 1 6 2 9 13 15 5
   4 21 16 12 3 14 11 22 18 19 7 10 1 17 8 6 2 20 9 13 15 5
   12 4 19 1 3 7 6 21 18 11 16 2 10 20 9 15 14 17 22 8 13 5
   4 21 3 16 14 11 12 22 18 10 19 20 17 8 7 6 1 2 13 15 9 5
   4 21 3 16 11 14 22 12 18 10 20 19 17 8 7 6 1 13 15 2 9 5
   4 3 14 11 21 12 16 22 19 10 18 20 17 1 7 8 2 13 9 6 15 5
   15 22 17 21 6 11 13 19 4 12 3 18 9 7 1 10 8 20 14 16 5 2
   12 4 3 7 21 19 1 18 11 6 16 2 14 10 17 22 20 9 15 8 13 5
   ;
```

The macro variable `&m` is set to 22, the number of combinations. This is done to make it easier to modify the code for future use with different sized studies. For each subject, the numbers of the 22 products are read into the variables `c1` through `c22`. The do loop, `do i = 1 to &m`, loops over each of the products. Consider the first product: `k` is set to `c[i]`, which is `c[1]`, which is 4 since the fourth product was ranked first by the first subject. The first data integrity check, `if 1 le k le &m then do` ensures that the number is in the valid range, 1 to 22. Otherwise an error is displayed. Since the number is valid, `r[k]` is checked to see if it is missing. If it is not missing, another error is displayed. The array `r` consists of 22 variables `r1` through 22. These variables start out each pass through the DATA step as missing and end up as the ranks. If `r[k] eq .`, then the *kth* combination has not had a rank assigned yet so everything is fine. If `r[k] ne .`, the same number appears twice in a subject's data so there is something wrong with the data entry. The statement `r[k] = i` assigns the ranks. For subject 1 and the first product, `k = c[i] = c[1] = 4` so the rank of the fourth product is set to 1 (`r[k] = r[4] = i = 1`). For subject 1 and the second product, `k = c[i] = c[2] = 3` so the rank of the third product is set to 2 (`r[k] = r[3] = i = 2`). For subject 1 and the last product, `k = c[i] = c[22] = 5` so the rank of the fifth product is set to 22 (`r[k] = r[5] = i = 22`). At the end of the `do i = 1 to &m` loop, each of the 22 variables in `r1-r22` should have been set to exactly one rank. If any of these variables are missing, then one or more product numbers did not appear in the data, so this is flagged as an error. The statement `name = 'Subj' || put(_n_, z2.)` creates a subject ID of the form `Subj01`, `Subj02`, ..., `Subj12`.

Say there was a mistake in data entry for the first subject—say product 17 had been entered as 7 instead of 17. We would get the following error messages:

```
ERROR: For subject 1, combination 7 is given more than once.
ERROR: For subject 1, combination 17 is not given.
```

If for the first subject, the 17 had been entered as 117 instead of 17, we would get the following error messages:

```
ERROR: For subject 1, combination 117 is invalid.
ERROR: For subject 1, combination 17 is not given.
```

The next step transposes the data set from one row per subject to one row per product. The `id name` statement in PROC TRANSPOSE names the rank variables `Subj01` through `Subj12`. Later, we will need to sort by these names. That is why we used leading zeros and names like `Subj01` instead of `Subj1`. Next, the input data set is merged with the design. The following steps process and display the data:

```
proc transpose data=ranks out=ranks2;
   id name;
   run;

data both;
   merge sasuser.dietdes ranks2;
   drop _name_;
   run;

proc print label;
   title2 'Data and Design Together';
   run;
```

The results are as follows:

```
                              Frozen Diet Entrees
                           Data and Design Together

Obs  Ingredient     Fat     Price  Calories  w  Subj01  Subj02  Subj03  Subj04

  1   Beef       2 Grams  $2.59     250     .      9       4       7       3
  2   Beef       5 Grams  $2.59     250     1     17      12      15      10
  3   Turkey     2 Grams  $1.99     350     1      2       3       2       8
  4   Turkey     2 Grams  $1.99     250     1      1       1       1       1
  5   Turkey     8 Grams  $2.59     350     1     22      21      22      21
  6   Chicken    2 Grams  $2.59     350     1      7       8       8      19
  7   Chicken    2 Grams  $2.59     250     1      3       6       3       9
  8   Chicken    8 Grams  $2.29     350     1     20      22      20      22
  9   Beef       5 Grams  $2.59     350     1     19      16      17      20
 10   Turkey     5 Grams  $2.29     250     1      6       7       9       4
 11   Beef       5 Grams  $1.99     350     .     12       9      12      14
 12   Beef       2 Grams  $2.29     250     1      5       2       4       2
 13   Turkey     8 Grams  $2.59     250     1     21      19      21      12
 14   Beef       8 Grams  $1.99     250     1     14      17      16       6
 15   Turkey     5 Grams  $2.59     350     .     16      15      14      18
 16   Chicken    5 Grams  $1.99     350     1     10      11      11      17
 17   Chicken    8 Grams  $2.29     250     1     15      20      18      11
 18   Chicken    5 Grams  $2.29     250     .     11      13      10       7
 19   Beef       2 Grams  $2.29     350     1      8       5       5      13
 20   Turkey     5 Grams  $2.29     350     1     13      14      13      16
 21   Chicken    5 Grams  $1.99     250     1      4      10       6       5
 22   Beef       8 Grams  $1.99     350     1     18      18      19      15

Obs  Subj05    Subj06    Subj07    Subj08    Subj09    Subj10    Subj11    Subj12

  1    16        13         4        17        17        14        15         7
  2    18        17        12        18        20        17        22        12
  3     6         5         5         3         3         2        11         3
  4     1         1         2         1         1         1         9         2
  5    22        22        22        22        22        22        21        22
  6    17        16         7        16        16        20         5        10
  7    15        11         6        15        15        15        14         4
  8    13        15        20        14        14        16        17        20
  9    19        19        15        21        21        19        13        18
 10    11        12        13        10        10        10        16        14
 11     4         7        10         6         5         4         6         9
 12     7         4         1         7         8         6        10         1
```

```
13    20        20        21        19        18        18         7        21
14     3         6        17         5         6         3        19        13
15    21        21        16        20        19        21         1        19
16     5         3        11         4         4         7        20        11
17    12        14        18        13        13        13         3        15
18    10         9         9         9         9        11        12         8
19     9        10         3        11        12         9         8         6
20    14        18        14        12        11        12        18        17
21     2         2         8         2         2         5         4         5
22     8         8        19         8         7         8         2        16
```

One more data set manipulation is sometimes necessary—the addition of *simulation* observations. Simulation observations are not rated by the subjects and do not contribute to the analysis. They are scored as passive observations. Simulations are *what-if* combinations. They are combinations that are entered to get a prediction of what their utility would have been if they had been rated. In this example, all combinations are added as simulations. The `%MktEx` macro is called to make a full-factorial design. The `n=` specification accepts expressions, so `n=3*3*3*2` and `n=54` are equivalent. The `data all` step reads in the design and data followed by the simulation observations. The flag variable `f` indicates when the simulation observations are being processed. Simulation observations are given a weight of 0 to exclude them from the analysis and to distinguish them from the holdouts. Notice that the dependent variable has missing values for the simulations and nonmissing values for the holdouts and active observations. The following steps process and display the design:

```
proc format;
   value wf 1 = 'Active'
            . = 'Holdout'
            0 = 'Simulation';
   run;

%mktex(3 3 3 2, n=3*3*3*2)
%mktlab(data=design, vars=Ingredient Fat Price Calories)

data all;
   set both final(in=f);
   if f then w = 0;
   format w wf.;
   run;

proc print data=all(Obs=25 drop=subj04-subj12) label;
   title2 'Some of the Final Data Set';
   run;
```

The data for the first three subjects and the first 25 rows of the data set are as follows:

```
                              Frozen Diet Entrees
                           Some of the Final Data Set

 Obs  Ingredient    Fat     Price  Calories  w           Subj01  Subj02  Subj03

   1   Beef       2 Grams  $2.59    250     Holdout         9       4       7
   2   Beef       5 Grams  $2.59    250     Active         17      12      15
   3   Turkey     2 Grams  $1.99    350     Active          2       3       2
   4   Turkey     2 Grams  $1.99    250     Active          1       1       1
   5   Turkey     8 Grams  $2.59    350     Active         22      21      22
   6   Chicken    2 Grams  $2.59    350     Active          7       8       8
   7   Chicken    2 Grams  $2.59    250     Active          3       6       3
   8   Chicken    8 Grams  $2.29    350     Active         20      22      20
   9   Beef       5 Grams  $2.59    350     Active         19      16      17
  10   Turkey     5 Grams  $2.29    250     Active          6       7       9
  11   Beef       5 Grams  $1.99    350     Holdout        12       9      12
  12   Beef       2 Grams  $2.29    250     Active          5       2       4
  13   Turkey     8 Grams  $2.59    250     Active         21      19      21
  14   Beef       8 Grams  $1.99    250     Active         14      17      16
  15   Turkey     5 Grams  $2.59    350     Holdout        16      15      14
  16   Chicken    5 Grams  $1.99    350     Active         10      11      11
  17   Chicken    8 Grams  $2.29    250     Active         15      20      18
  18   Chicken    5 Grams  $2.29    250     Holdout        11      13      10
  19   Beef       2 Grams  $2.29    350     Active          8       5       5
  20   Turkey     5 Grams  $2.29    350     Active         13      14      13
  21   Chicken    5 Grams  $1.99    250     Active          4      10       6
  22   Beef       8 Grams  $1.99    350     Active         18      18      19
  23   Chicken    8 Grams  $2.59    350     Simulation      .       .       .
  24   Chicken    8 Grams  $2.59    350     Simulation      .       .       .
  25   Chicken    8 Grams  $2.59    350     Simulation      .       .       .
```

Metric Conjoint Analysis

In this part of this example, PROC TRANSREG performs the conjoint analysis as follows:

```
ods exclude notes mvanova anova;
proc transreg data=all utilities short separators=', '
   method=morals outtest=utils;
   title2 'Conjoint Analysis';
   model identity(subj: / reflect) =
         class(Ingredient Fat Price Calories / zero=sum);
   weight w;
   output p ireplace out=results coefficients;
   run;
```

The `proc`, `model`, and `output` statements are typical for a conjoint analysis of rank-order data with more than one subject. (In this analysis, we perform a metric conjoint analysis. It is more typical to perform nonmetric conjoint analysis of rank-order data. However, it is not absolutely required.) The `proc` statement specifies `method=morals`, which fits the conjoint analysis model separately for each subject. The `proc` statement also requests an `outtest=` data set, which contains the ANOVA and part-worth utilities tables from the displayed output. In the `model` statement, the dependent variable list `subj:` specifies all variables in the DATA= data set that begin with the prefix `subj` (in this case `subj01-subj12`). The `weight` variable designates the active (`weight` = 1), holdout (`weight` = .), and simulation (`weight` = 0) observations. Only the active observations are used to compute the part-worth utilities. However, predicted utilities are computed for all observations, including active, holdouts, and simulations, using those part-worths. The `output` statement creates an `out=` data set beginning with all results for the first subject, followed by all subject two results, and so on.

Conjoint analysis fits individual-level models. There is one set of output for each subject. The results are as follows:

```
                              Frozen Diet Entrees
                               Conjoint Analysis

                            The TRANSREG Procedure

                           Class Level Information

               Class         Levels    Values

               Ingredient         3    Chicken  Beef  Turkey

               Fat                3    8 Grams  5 Grams  2 Grams

               Price              3    $2.59  $2.29  $1.99

               Calories           2    350  250

                   Number of Observations Read            76
                   Number of Observations Used            18
                   Sum of Weights Read                    18
                   Sum of Weights Used                    18
```

```
                          Frozen Diet Entrees
                           Conjoint Analysis

                         The TRANSREG Procedure

Identity(Subj01)
Algorithm converged.


       The TRANSREG Procedure Hypothesis Tests for Identity(Subj01)


             Root MSE           1.81046    R-Square    0.9618
             Dependent Mean    11.38889    Adj R-Sq    0.9351
             Coeff Var         15.89675


                          Part-Worth Utilities

                                                          Importance
                                             Standard    (% Utility
         Label                     Utility      Error        Range)

         Intercept                 11.3889    0.42673

         Ingredient, Chicken        1.5556    0.60349        13.095
         Ingredient, Beef          -2.1111    0.60349
         Ingredient, Turkey         0.5556    0.60349

         Fat, 8 Grams              -6.9444    0.60349        50.000
         Fat, 5 Grams              -0.1111    0.60349
         Fat, 2 Grams               7.0556    0.60349

         Price, $2.59              -3.4444    0.60349        23.810
         Price, $2.29               0.2222    0.60349
         Price, $1.99               3.2222    0.60349

         Calories, 350             -1.8333    0.42673        13.095
         Calories, 250              1.8333    0.42673
```

```
                                Frozen Diet Entrees
                                 Conjoint Analysis

                               The TRANSREG Procedure

      Identity(Subj02)
      Algorithm converged.


             The TRANSREG Procedure Hypothesis Tests for Identity(Subj02)


                  Root MSE            1.30809    R-Square    0.9788
                  Dependent Mean     11.77778    Adj R-Sq    0.9640
                  Coeff Var          11.10646


                                Part-Worth Utilities

                                                            Importance
                                                 Standard   (% Utility
              Label                   Utility       Error       Range)

              Intercept               11.7778     0.30832

              Ingredient, Chicken     -1.0556     0.43603        8.451
              Ingredient, Beef         0.1111     0.43603
              Ingredient, Turkey       0.9444     0.43603

              Fat, 8 Grams            -7.7222     0.43603       64.789
              Fat, 5 Grams             0.1111     0.43603
              Fat, 2 Grams             7.6111     0.43603

              Price, $2.59            -1.8889     0.43603       15.493
              Price, $2.29             0.1111     0.43603
              Price, $1.99             1.7778     0.43603

              Calories, 350           -1.3333     0.30832       11.268
              Calories, 250            1.3333     0.30832
```

```
                              Frozen Diet Entrees
                               Conjoint Analysis

                            The TRANSREG Procedure

   Identity(Subj03)
   Algorithm converged.


          The TRANSREG Procedure Hypothesis Tests for Identity(Subj03)


                Root MSE           1.15470    R-Square    0.9844
                Dependent Mean    11.66667    Adj R-Sq    0.9735
                Coeff Var          9.89743


                            Part-Worth Utilities

                                                            Importance
                                                 Standard    (% Utility
            Label                    Utility        Error        Range)

            Intercept                11.6667      0.27217

            Ingredient, Chicken       0.6667      0.38490         6.667
            Ingredient, Beef         -1.0000      0.38490
            Ingredient, Turkey        0.3333      0.38490

            Fat, 8 Grams             -7.6667      0.38490        62.000
            Fat, 5 Grams             -0.1667      0.38490
            Fat, 2 Grams              7.8333      0.38490

            Price, $2.59             -2.6667      0.38490        20.667
            Price, $2.29              0.1667      0.38490
            Price, $1.99              2.5000      0.38490

            Calories, 350            -1.3333      0.27217        10.667
            Calories, 250             1.3333      0.27217
```

```
                              Frozen Diet Entrees
                               Conjoint Analysis

                            The TRANSREG Procedure

       Identity(Subj04)
       Algorithm converged.


              The TRANSREG Procedure Hypothesis Tests for Identity(Subj04)


                    Root MSE             1.05935    R-Square    0.9849
                    Dependent Mean      11.72222    Adj R-Sq    0.9743
                    Coeff Var            9.03711


                              Part-Worth Utilities

                                                                Importance
                                                    Standard    (% Utility
                Label                    Utility       Error        Range)

                Intercept                11.7222     0.24969

                Ingredient, Chicken      -2.1111     0.35312        13.490
                Ingredient, Beef          0.7222     0.35312
                Ingredient, Turkey        1.3889     0.35312

                Fat, 8 Grams             -2.7778     0.35312        22.484
                Fat, 5 Grams             -0.2778     0.35312
                Fat, 2 Grams              3.0556     0.35312

                Price, $2.59             -3.4444     0.35312        25.054
                Price, $2.29              0.3889     0.35312
                Price, $1.99              3.0556     0.35312

                Calories, 350            -5.0556     0.24969        38.972
                Calories, 250             5.0556     0.24969
```

```
                              Frozen Diet Entrees
                               Conjoint Analysis

                            The TRANSREG Procedure

      Identity(Subj05)
      Algorithm converged.


             The TRANSREG Procedure Hypothesis Tests for Identity(Subj05)


                   Root MSE           1.02198    R-Square    0.9854
                   Dependent Mean    11.22222    Adj R-Sq    0.9752
                   Coeff Var          9.10676


                              Part-Worth Utilities

                                                            Importance
                                                 Standard   (% Utility
               Label                   Utility      Error       Range)

               Intercept               11.2222    0.24088

               Ingredient, Chicken      0.5556    0.34066        7.407
               Ingredient, Beef         0.5556    0.34066
               Ingredient, Turkey      -1.1111    0.34066

               Fat, 8 Grams            -1.7778    0.34066       17.037
               Fat, 5 Grams            -0.2778    0.34066
               Fat, 2 Grams             2.0556    0.34066

               Price, $2.59            -7.2778    0.34066       63.704
               Price, $2.29             0.2222    0.34066
               Price, $1.99             7.0556    0.34066

               Calories, 350           -1.3333    0.24088       11.852
               Calories, 250            1.3333    0.24088
```

```
                              Frozen Diet Entrees
                               Conjoint Analysis

                            The TRANSREG Procedure

       Identity(Subj06)
       Algorithm converged.


              The TRANSREG Procedure Hypothesis Tests for Identity(Subj06)


                    Root MSE           1.67000    R-Square    0.9636
                    Dependent Mean    11.27778    Adj R-Sq    0.9381
                    Coeff Var         14.80785


                              Part-Worth Utilities

                                                             Importance
                                                  Standard   (% Utility
                Label                   Utility      Error       Range)

                Intercept               11.2778    0.39362

                Ingredient, Chicken      1.1111    0.55667       11.015
                Ingredient, Beef         0.6111    0.55667
                Ingredient, Turkey      -1.7222    0.55667

                Fat, 8 Grams            -2.8889    0.55667       24.622
                Fat, 5 Grams            -0.5556    0.55667
                Fat, 2 Grams             3.4444    0.55667

                Price, $2.59            -6.2222    0.55667       51.836
                Price, $2.29            -0.8889    0.55667
                Price, $1.99             7.1111    0.55667

                Calories, 350           -1.6111    0.39362       12.527
                Calories, 250            1.6111    0.39362
```

```
                              Frozen Diet Entrees
                               Conjoint Analysis

                            The TRANSREG Procedure

      Identity(Subj07)
      Algorithm converged.


             The TRANSREG Procedure Hypothesis Tests for Identity(Subj07)


                   Root MSE           1.06979    R-Square    0.9857
                   Dependent Mean    11.88889    Adj R-Sq    0.9756
                   Coeff Var          8.99821


                              Part-Worth Utilities

                                                              Importance
                                                  Standard    (% Utility
               Label                   Utility       Error        Range)

               Intercept               11.8889     0.25215

               Ingredient, Chicken      0.2222     0.35660         7.353
               Ingredient, Beef         0.7222     0.35660
               Ingredient, Turkey      -0.9444     0.35660

               Fat, 8 Grams            -7.6111     0.35660        68.382
               Fat, 5 Grams            -0.2778     0.35660
               Fat, 2 Grams             7.8889     0.35660

               Price, $2.59            -1.9444     0.35660        15.441
               Price, $2.29             0.3889     0.35660
               Price, $1.99             1.5556     0.35660

               Calories, 350           -1.0000     0.25215         8.824
               Calories, 250            1.0000     0.25215
```

```
                          Frozen Diet Entrees
                           Conjoint Analysis

                         The TRANSREG Procedure

Identity(Subj08)
Algorithm converged.


       The TRANSREG Procedure Hypothesis Tests for Identity(Subj08)


            Root MSE           0.79582    R-Square    0.9915
            Dependent Mean    11.16667    Adj R-Sq    0.9855
            Coeff Var          7.12677


                          Part-Worth Utilities

                                                        Importance
                                            Standard    (% Utility
        Label                    Utility       Error        Range)

        Intercept                11.1667     0.18758

        Ingredient, Chicken       0.5000     0.26527         4.412
        Ingredient, Beef         -0.5000     0.26527
        Ingredient, Turkey        0.0000     0.26527

        Fat, 8 Grams             -2.3333     0.26527        20.588
        Fat, 5 Grams              0.0000     0.26527
        Fat, 2 Grams              2.3333     0.26527

        Price, $2.59             -7.3333     0.26527        64.706
        Price, $2.29             -0.0000     0.26527
        Price, $1.99              7.3333     0.26527

        Calories, 350            -1.1667     0.18758        10.294
        Calories, 250             1.1667     0.18758
```

```
                              Frozen Diet Entrees
                               Conjoint Analysis

                            The TRANSREG Procedure

      Identity(Subj09)
      Algorithm converged.


             The TRANSREG Procedure Hypothesis Tests for Identity(Subj09)


                   Root MSE           1.05935    R-Square    0.9850
                   Dependent Mean    11.27778    Adj R-Sq    0.9745
                   Coeff Var          9.39325


                              Part-Worth Utilities

                                                             Importance
                                                Standard    (% Utility
               Label                  Utility      Error        Range)

               Intercept              11.2778    0.24969

               Ingredient, Chicken     0.6111    0.35312         7.389
               Ingredient, Beef       -1.0556    0.35312
               Ingredient, Turkey      0.4444    0.35312

               Fat, 8 Grams           -2.0556    0.35312        18.473
               Fat, 5 Grams           -0.0556    0.35312
               Fat, 2 Grams            2.1111    0.35312

               Price, $2.59           -7.3889    0.35312        65.764
               Price, $2.29           -0.0556    0.35312
               Price, $1.99            7.4444    0.35312

               Calories, 350          -0.9444    0.24969         8.374
               Calories, 250           0.9444    0.24969
```

```
                         Frozen Diet Entrees
                          Conjoint Analysis

                        The TRANSREG Procedure

Identity(Subj10)
Algorithm converged.


       The TRANSREG Procedure Hypothesis Tests for Identity(Subj10)


             Root MSE           0.90062    R-Square    0.9889
             Dependent Mean    11.27778    Adj R-Sq    0.9812
             Coeff Var          7.98577


                         Part-Worth Utilities

                                                        Importance
                                             Standard   (% Utility
         Label                    Utility       Error       Range)

         Intercept                11.2778     0.21228

         Ingredient, Chicken      -1.3889     0.30021        9.722
         Ingredient, Beef          0.9444     0.30021
         Ingredient, Turkey        0.4444     0.30021

         Fat, 8 Grams             -2.0556     0.30021       18.750
         Fat, 5 Grams             -0.3889     0.30021
         Fat, 2 Grams              2.4444     0.30021

         Price, $2.59             -7.2222     0.30021       59.028
         Price, $2.29              0.2778     0.30021
         Price, $1.99              6.9444     0.30021

         Calories, 350            -1.5000     0.21228       12.500
         Calories, 250             1.5000     0.21228
```

```
                              Frozen Diet Entrees
                               Conjoint Analysis

                            The TRANSREG Procedure

      Identity(Subj11)
      Algorithm converged.


             The TRANSREG Procedure Hypothesis Tests for Identity(Subj11)


                   Root MSE           7.42369    R-Square    0.2393
                   Dependent Mean    12.16667    Adj R-Sq   -0.2932
                   Coeff Var         61.01660


                              Part-Worth Utilities

                                                                Importance
                                                     Standard   (% Utility
               Label                      Utility       Error       Range)

               Intercept                  12.1667     1.74978

               Ingredient, Chicken         1.6667     2.47456       23.950
               Ingredient, Beef           -0.1667     2.47456
               Ingredient, Turkey         -1.5000     2.47456

               Fat, 8 Grams                0.6667     2.47456       45.378
               Fat, 5 Grams               -3.3333     2.47456
               Fat, 2 Grams                2.6667     2.47456

               Price, $2.59               -1.5000     2.47456       21.429
               Price, $2.29                0.1667     2.47456
               Price, $1.99                1.3333     2.47456

               Calories, 350              -0.6111     1.74978        9.244
               Calories, 250               0.6111     1.74978
```

```
                                Frozen Diet Entrees
                                 Conjoint Analysis

                              The TRANSREG Procedure

      Identity(Subj12)
      Algorithm converged.


             The TRANSREG Procedure Hypothesis Tests for Identity(Subj12)


                   Root MSE           1.49443    R-Square    0.9717
                   Dependent Mean    11.66667    Adj R-Sq    0.9519
                   Coeff Var         12.80944


                              Part-Worth Utilities

                                                                Importance
                                                     Standard   (% Utility
               Label                     Utility        Error       Range)

               Intercept                 11.6667      0.35224

               Ingredient, Chicken        0.8333      0.49814        8.974
               Ingredient, Beef           0.6667      0.49814
               Ingredient, Turkey        -1.5000      0.49814

               Fat, 8 Grams              -6.1667      0.49814       51.923
               Fat, 5 Grams              -1.1667      0.49814
               Fat, 2 Grams               7.3333      0.49814

               Price, $2.59              -2.8333      0.49814       23.718
               Price, $2.29              -0.5000      0.49814
               Price, $1.99               3.3333      0.49814

               Calories, 350             -2.0000      0.35224       15.385
               Calories, 250              2.0000      0.35224
```

Recall that we used an `ods exclude` statement and we used PROC TEMPLATE on page 683 to customize the output from PROC TRANSREG.

The following step displays some of the output data set to see the predicted utilities for the first two subjects:

```
proc print data=results(drop=_depend_ t_depend_ intercept &_trgind) label;
   title2 'Predicted Utility';
   where w ne 0 and _depvar_ le 'Identity(Subj02)' and not (_type_ =: 'M');
   by _depvar_;
   label p_depend_ = 'Predicted Utility';
   run;
```

We display _TYPE_, _NAME_, and the weight variable, w; drop the original and transformed dependent variable, _depend_ and t_depend_; display the predicted values (predicted utilities), p_depend_; drop the intercept and coded independent variables; and display the original class variables. Note that the macro variable &_trgind is automatically created by PROC TRANSREG and its value is a list of the names of the coded variables. The where statement is used to exclude the simulation observations and just show results for the first two subjects. The predicted utilities for each of the rated products for the first two subjects are as follows:

```
                              Frozen Diet Entrees
                               Predicted Utility

  ----------- Dependent Variable Transformation(Name)=Identity(Subj01) -----------

                                  Predicted
 Obs  _TYPE_  _NAME_     w         Utility   Ingredient    Fat    Price  Calories

   1          ROW1    Holdout    14.7222   Beef       2 Grams  $2.59    250
   2  SCORE   ROW2    Active      7.5556   Beef       5 Grams  $2.59    250
   3  SCORE   ROW3    Active     20.3889   Turkey     2 Grams  $1.99    350
   4  SCORE   ROW4    Active     24.0556   Turkey     2 Grams  $1.99    250
   5  SCORE   ROW5    Active     -0.2778   Turkey     8 Grams  $2.59    350
   6  SCORE   ROW6    Active     14.7222   Chicken    2 Grams  $2.59    350
   7  SCORE   ROW7    Active     18.3889   Chicken    2 Grams  $2.59    250
   8  SCORE   ROW8    Active      4.3889   Chicken    8 Grams  $2.29    350
   9  SCORE   ROW9    Active      3.8889   Beef       5 Grams  $2.59    350
  10  SCORE   ROW10   Active     13.8889   Turkey     5 Grams  $2.29    250
  11          ROW11   Holdout    10.5556   Beef       5 Grams  $1.99    350
  12  SCORE   ROW12   Active     18.3889   Beef       2 Grams  $2.29    250
  13  SCORE   ROW13   Active      3.3889   Turkey     8 Grams  $2.59    250
  14  SCORE   ROW14   Active      7.3889   Beef       8 Grams  $1.99    250
  15          ROW15   Holdout     6.5556   Turkey     5 Grams  $2.59    350
  16  SCORE   ROW16   Active     14.2222   Chicken    5 Grams  $1.99    350
  17  SCORE   ROW17   Active      8.0556   Chicken    8 Grams  $2.29    250
  18          ROW18   Holdout    14.8889   Chicken    5 Grams  $2.29    250
  19  SCORE   ROW19   Active     14.7222   Beef       2 Grams  $2.29    350
  20  SCORE   ROW20   Active     10.2222   Turkey     5 Grams  $2.29    350
  21  SCORE   ROW21   Active     17.8889   Chicken    5 Grams  $1.99    250
  22  SCORE   ROW22   Active      3.7222   Beef       8 Grams  $1.99    350
```

```
    ----------- Dependent Variable Transformation(Name)=Identity(Subj02) -----------

                                     Predicted
     Obs  _TYPE_  _NAME_     w        Utility   Ingredient    Fat    Price  Calories

      79          ROW1    Holdout   18.9444    Beef       2 Grams  $2.59    250
      80  SCORE   ROW2    Active    11.4444    Beef       5 Grams  $2.59    250
      81  SCORE   ROW3    Active    20.7778    Turkey     2 Grams  $1.99    350
      82  SCORE   ROW4    Active    23.4444    Turkey     2 Grams  $1.99    250
      83  SCORE   ROW5    Active     1.7778    Turkey     8 Grams  $2.59    350
      84  SCORE   ROW6    Active    15.1111    Chicken    2 Grams  $2.59    350
      85  SCORE   ROW7    Active    17.7778    Chicken    2 Grams  $2.59    250
      86  SCORE   ROW8    Active     1.7778    Chicken    8 Grams  $2.29    350
      87  SCORE   ROW9    Active     8.7778    Beef       5 Grams  $2.59    350
      88  SCORE   ROW10   Active    14.2778    Turkey     5 Grams  $2.29    250
      89          ROW11   Holdout   12.4444    Beef       5 Grams  $1.99    350
      90  SCORE   ROW12   Active    20.9444    Beef       2 Grams  $2.29    250
      91  SCORE   ROW13   Active     4.4444    Turkey     8 Grams  $2.59    250
      92  SCORE   ROW14   Active     7.2778    Beef       8 Grams  $1.99    250
      93          ROW15   Holdout    9.6111    Turkey     5 Grams  $2.59    350
      94  SCORE   ROW16   Active    11.2778    Chicken    5 Grams  $1.99    350
      95  SCORE   ROW17   Active     4.4444    Chicken    8 Grams  $2.29    250
      96          ROW18   Holdout   12.2778    Chicken    5 Grams  $2.29    250
      97  SCORE   ROW19   Active    18.2778    Beef       2 Grams  $2.29    350
      98  SCORE   ROW20   Active    11.6111    Turkey     5 Grams  $2.29    350
      99  SCORE   ROW21   Active    13.9444    Chicken    5 Grams  $1.99    250
     100  SCORE   ROW22   Active     4.6111    Beef       8 Grams  $1.99    350
```

Analyzing Holdouts

The next steps display the correlations between the predicted utility for holdout observations and their actual ratings. These correlations provide a measure of the validity of the results, since the holdout observations have zero weight and do not contribute to any of the calculations. The Pearson correlations are the ordinary correlation coefficients, and the Kendall Tau's are rank-based measures of correlation. These correlations should always be large. Subjects whose correlations are small may be unreliable.

PROC CORR is used to produce the correlations. Since the output is not very compact, ODS is used to suppress the normal displayed output (`ods listing close`), output the Pearson correlations to an output data set P (`PearsonCorr=p`), and output the Kendall correlations to an output data set K (`KendallCorr=k`). The listing is reopened for normal output (`ods listing`), the two tables are merged renaming the variables to identify the correlation type, the subject number is pulled out of the subject variable names, and the results are displayed. The following steps perform the analysis and display the results:

```
ods output KendallCorr=k PearsonCorr=p;
ods listing close;
proc corr nosimple noprob kendall pearson
   data=results(where=(w=.));
   title2 'Holdout Validation Results';
   var p_depend_;
   with t_depend_;
   by notsorted _depvar_;
   run;
ods listing;

data both(keep=subject pearson kendall);
   length Subject 8;
   merge p(rename=(p_depend_=Pearson))
         k(rename=(p_depend_=Kendall));
   subject = input(substr(_depvar_, 14, 2), best2.);
   run;

proc print; run;
```

The results are as follows:

```
                          Frozen Diet Entrees
                      Holdout Validation Results

                 Obs    Subject     Pearson     Kendall

                   1        1       0.93848     0.66667
                   2        2       0.94340     1.00000
                   3        3       0.99038     1.00000
                   4        4       0.97980     1.00000
                   5        5       0.98930     1.00000
                   6        6       0.98649     1.00000
                   7        7       0.99029     1.00000
                   8        8       0.99296     1.00000
                   9        9       0.99873     1.00000
                  10       10       0.99973     1.00000
                  11       11      -0.98184    -1.00000
                  12       12       0.92920     1.00000
```

Most of the correlations look great! However, the results from subject 11 look suspect. Subject 11's holdout correlations are negative. We can return to page 734 and look at the conjoint results. Subject 11 has an R square of 0.2393. In contrast, all of the other subjects have an R square over 0.95. Subject 11 almost certainly did not take the task seriously, so his or her results need to be discarded. The following steps discard the results from Subject 11:

```
data results2;
   set results;
   if not (index(_depvar_, '11'));
   run;

data utils2;
   set utils;
   if not (index(_depvar_, '11'));
   run;
```

Simulations

The next steps display simulation observations. The most preferred combinations are displayed for each subject as follows:

```
proc sort data=results2(where=(w=0)) out=sims(drop=&_trgind);
   by _depvar_ descending p_depend_;
   run;

data sims; /* Pull out first 10 for each subject. */
   set sims;
   by _depvar_;
   retain n 0;
   if first._depvar_ then n = 0;
   n = n + 1;
   if n le 10;
   drop w _depend_ t_depend_ n _name_ _type_ intercept;
   run;

proc print data=sims label;
   by _depvar_ ;
   title2 'Simulations Sorted by Decreasing Predicted Utility';
   title3 'Just the Ten Most Preferred Combinations are Printed';
   label p_depend_ = 'Predicted Utility';
   run;
```

The results are as follows:

```
                          Frozen Diet Entrees
                 Simulations Sorted by Decreasing Predicted Utility
                Just the Ten Most Preferred Combinations are Printed

----------- Dependent Variable Transformation(Name)=Identity(Subj01) -----------

              Predicted
        Obs     Utility     Ingredient      Fat      Price    Calories

          1     22.0556      Chicken      2 Grams    $2.29       250
          2     22.0556      Chicken      2 Grams    $2.29       250
          3     22.0556      Chicken      2 Grams    $2.29       250
          4     21.3889      Chicken      2 Grams    $1.99       350
          5     21.3889      Chicken      2 Grams    $1.99       350
          6     21.3889      Chicken      2 Grams    $1.99       350
          7     20.3889      Turkey       2 Grams    $1.99       350
          8     20.3889      Turkey       2 Grams    $1.99       350
          9     20.3889      Turkey       2 Grams    $1.99       350
         10     18.3889      Beef         2 Grams    $2.29       250

----------- Dependent Variable Transformation(Name)=Identity(Subj02) -----------

              Predicted
        Obs     Utility     Ingredient      Fat      Price    Calories

         11     20.9444      Beef         2 Grams    $2.29       250
         12     20.9444      Beef         2 Grams    $2.29       250
         13     20.9444      Beef         2 Grams    $2.29       250
         14     20.7778      Turkey       2 Grams    $1.99       350
         15     20.7778      Turkey       2 Grams    $1.99       350
         16     20.7778      Turkey       2 Grams    $1.99       350
         17     19.7778      Chicken      2 Grams    $2.29       250
         18     19.7778      Chicken      2 Grams    $2.29       250
         19     19.7778      Chicken      2 Grams    $2.29       250
         20     19.7778      Turkey       2 Grams    $2.59       250

----------- Dependent Variable Transformation(Name)=Identity(Subj03) -----------

              Predicted
        Obs     Utility     Ingredient      Fat      Price    Calories

         21     21.6667      Chicken      2 Grams    $2.29       250
         22     21.6667      Chicken      2 Grams    $2.29       250
         23     21.6667      Chicken      2 Grams    $2.29       250
         24     21.3333      Chicken      2 Grams    $1.99       350
         25     21.3333      Chicken      2 Grams    $1.99       350
```

```
          26     21.3333      Chicken      2 Grams    $1.99      350
          27     21.0000      Turkey       2 Grams    $1.99      350
          28     21.0000      Turkey       2 Grams    $1.99      350
          29     21.0000      Turkey       2 Grams    $1.99      350
          30     20.0000      Beef         2 Grams    $2.29      250

----------- Dependent Variable Transformation(Name)=Identity(Subj04) -----------

              Predicted
         Obs    Utility     Ingredient      Fat       Price    Calories

          31     20.9444      Beef         2 Grams    $2.29      250
          32     20.9444      Beef         2 Grams    $2.29      250
          33     20.9444      Beef         2 Grams    $2.29      250
          34     20.2778      Beef         5 Grams    $1.99      250
          35     20.2778      Beef         5 Grams    $1.99      250
          36     20.2778      Beef         5 Grams    $1.99      250
          37     18.4444      Turkey       8 Grams    $1.99      250
          38     18.4444      Turkey       8 Grams    $1.99      250
          39     18.4444      Turkey       8 Grams    $1.99      250
          40     18.1111      Chicken      2 Grams    $2.29      250

----------- Dependent Variable Transformation(Name)=Identity(Subj05) -----------

              Predicted
         Obs    Utility     Ingredient      Fat       Price    Calories

          41     19.8889      Beef         5 Grams    $1.99      250
          42     19.8889      Beef         5 Grams    $1.99      250
          43     19.8889      Beef         5 Grams    $1.99      250
          44     19.5556      Chicken      2 Grams    $1.99      350
          45     19.5556      Chicken      2 Grams    $1.99      350
          46     19.5556      Chicken      2 Grams    $1.99      350
          47     18.3889      Beef         8 Grams    $1.99      250
          48     18.3889      Beef         8 Grams    $1.99      250
          49     18.3889      Beef         8 Grams    $1.99      250
          50     17.8889      Turkey       2 Grams    $1.99      350

----------- Dependent Variable Transformation(Name)=Identity(Subj06) -----------

              Predicted
         Obs    Utility     Ingredient      Fat       Price    Calories

          51     21.3333      Chicken      2 Grams    $1.99      350
          52     21.3333      Chicken      2 Grams    $1.99      350
          53     21.3333      Chicken      2 Grams    $1.99      350
          54     20.0556      Beef         5 Grams    $1.99      250
          55     20.0556      Beef         5 Grams    $1.99      250
```

```
            56     20.0556      Beef          5 Grams    $1.99      250
            57     18.5000      Turkey        2 Grams    $1.99      350
            58     18.5000      Turkey        2 Grams    $1.99      350
            59     18.5000      Turkey        2 Grams    $1.99      350
            60     17.7222      Beef          8 Grams    $1.99      250

----------- Dependent Variable Transformation(Name)=Identity(Subj07) -----------

                Predicted
           Obs    Utility     Ingredient      Fat      Price    Calories

            61     21.8889      Beef          2 Grams    $2.29      250
            62     21.8889      Beef          2 Grams    $2.29      250
            63     21.8889      Beef          2 Grams    $2.29      250
            64     21.3889      Chicken       2 Grams    $2.29      250
            65     21.3889      Chicken       2 Grams    $2.29      250
            66     21.3889      Chicken       2 Grams    $2.29      250
            67     20.5556      Chicken       2 Grams    $1.99      350
            68     20.5556      Chicken       2 Grams    $1.99      350
            69     20.5556      Chicken       2 Grams    $1.99      350
            70     19.3889      Turkey        2 Grams    $1.99      350

----------- Dependent Variable Transformation(Name)=Identity(Subj08) -----------

                Predicted
           Obs    Utility     Ingredient      Fat      Price    Calories

            71     20.1667      Chicken       2 Grams    $1.99      350
            72     20.1667      Chicken       2 Grams    $1.99      350
            73     20.1667      Chicken       2 Grams    $1.99      350
            74     19.6667      Turkey        2 Grams    $1.99      350
            75     19.6667      Turkey        2 Grams    $1.99      350
            76     19.6667      Turkey        2 Grams    $1.99      350
            77     19.1667      Beef          5 Grams    $1.99      250
            78     19.1667      Beef          5 Grams    $1.99      250
            79     19.1667      Beef          5 Grams    $1.99      250
            80     17.8333      Chicken       5 Grams    $1.99      350

----------- Dependent Variable Transformation(Name)=Identity(Subj09) -----------

                Predicted
           Obs    Utility     Ingredient      Fat      Price    Calories

            81     20.5000      Chicken       2 Grams    $1.99      350
            82     20.5000      Chicken       2 Grams    $1.99      350
            83     20.5000      Chicken       2 Grams    $1.99      350
            84     20.3333      Turkey        2 Grams    $1.99      350
            85     20.3333      Turkey        2 Grams    $1.99      350
```

```
          86      20.3333      Turkey       2 Grams    $1.99       350
          87      18.5556      Beef         5 Grams    $1.99       250
          88      18.5556      Beef         5 Grams    $1.99       250
          89      18.5556      Beef         5 Grams    $1.99       250
          90      18.3333      Chicken      5 Grams    $1.99       350

---------- Dependent Variable Transformation(Name)=Identity(Subj10) ----------

                Predicted
         Obs     Utility     Ingredient      Fat       Price    Calories

          91      20.2778       Beef        5 Grams    $1.99       250
          92      20.2778       Beef        5 Grams    $1.99       250
          93      20.2778       Beef        5 Grams    $1.99       250
          94      19.6111       Turkey      2 Grams    $1.99       350
          95      19.6111       Turkey      2 Grams    $1.99       350
          96      19.6111       Turkey      2 Grams    $1.99       350
          97      18.6111       Beef        8 Grams    $1.99       250
          98      18.6111       Beef        8 Grams    $1.99       250
          99      18.6111       Beef        8 Grams    $1.99       250
         100      18.1111       Turkey      8 Grams    $1.99       250

---------- Dependent Variable Transformation(Name)=Identity(Subj12) ----------

                Predicted
         Obs     Utility     Ingredient      Fat       Price    Calories

         101      21.3333      Chicken      2 Grams    $2.29       250
         102      21.3333      Chicken      2 Grams    $2.29       250
         103      21.3333      Chicken      2 Grams    $2.29       250
         104      21.1667      Chicken      2 Grams    $1.99       350
         105      21.1667      Chicken      2 Grams    $1.99       350
         106      21.1667      Chicken      2 Grams    $1.99       350
         107      21.1667      Beef         2 Grams    $2.29       250
         108      21.1667      Beef         2 Grams    $2.29       250
         109      21.1667      Beef         2 Grams    $2.29       250
         110      18.8333      Turkey       2 Grams    $1.99       350
```

Summarizing Results Across Subjects

Conjoint analyses are performed on an individual basis, but usually the goal is to summarize the results across subjects. The `outtest=` data set contains all of the information in the displayed output and can be manipulated to create additional reports including a list of the individual R squares and the average of the importance values across subjects. The following step lists the variables in the `outtest=` data set:

```
proc contents data=utils2 position;
   ods select position;
   title2 'Variables in the OUTTEST= Data Set';
   run;
```

The results are as follows:

```
                              Frozen Diet Entrees
                       Variables in the OUTTEST= Data Set

                            The CONTENTS Procedure

                         Variables in Creation Order

 #    Variable       Type    Len    Label

 1    _DEPVAR_       Char     42    Dependent Variable Transformation(Name)
 2    _TYPE_         Char      8
 3    Title          Char     80    Title
 4    Variable       Char     42    Variable
 5    Coefficient    Num       8    Coefficient
 6    Statistic      Char     24    Statistic
 7    Value          Num       8    Value
 8    NumDF          Num       8    Num DF
 9    DenDF          Num       8    Den DF
10    SSq            Num       8    Sum of Squares
11    MeanSquare     Num       8    Mean Square
12    F              Num       8    F Value
13    NumericP       Num       8    Numeric (Approximate) p Value
14    P              Char      9    Formatted p Value
15    LowerLimit     Num       8    95% Lower Confidence Limit
16    UpperLimit     Num       8    95% Upper Confidence Limit
17    StdError       Num       8    Standard Error
18    Importance     Num       8    Importance (% Utility Range)
19    Label          Char    256    Label
```

The individual R squares are displayed in the `Value` variable for observations whose `Statistic` value is "R-Square" as follows:

```
proc print data=utils2 label;
   title2 'R-Squares';
   id _depvar_;
   var value;
   format value 4.2;
   where statistic = 'R-Square';
   label value = 'R-Square' _depvar_ = 'Subject';
   run;
```

The results are as follows:

```
                              Frozen Diet Entrees
                                   R-Squares

                              Subject         R-Square

                          Identity(Subj01)      0.96
                          Identity(Subj02)      0.98
                          Identity(Subj03)      0.98
                          Identity(Subj04)      0.98
                          Identity(Subj05)      0.99
                          Identity(Subj06)      0.96
                          Identity(Subj07)      0.99
                          Identity(Subj08)      0.99
                          Identity(Subj09)      0.99
                          Identity(Subj10)      0.99
                          Identity(Subj12)      0.97
```

The next steps extract the importance values and create a table. The DATA step extracts the importance values and creates row and column labels. The PROC TRANSPOSE step creates a subjects by attributes matrix from a vector (of the number of subjects times the number of attribute values). PROC PRINT displays the importance values, and PROC MEANS displays the average importances as follows:

```
data im;
   set utils2;
   if n(importance);   /* Exclude all missing, including specials.*/
   _depvar_ = scan(_depvar_, 2);   /* Discard transformation.     */
   label    = scan(label, 1, ','); /* Use up to comma for label.  */
   keep importance _depvar_ label;
   run;

proc transpose data=im out=im(drop=_name_ _label_);
   id label;
   by notsorted _depvar_;
   var importance;
   label _depvar_ = 'Subject';
   run;

proc print label;
   title2 'Importances';
   format _numeric_ 2.;
   id _depvar_;
   run;

proc means mean;
   title2 'Average Importances';
   run;
```

The results are as follows:

```
                              Frozen Diet Entrees
                                  Importances

                 Subject    Ingredient    Fat    Price    Calories

                 Subj01         13         50      24         13
                 Subj02          8         65      15         11
                 Subj03          7         62      21         11
                 Subj04         13         22      25         39
                 Subj05          7         17      64         12
                 Subj06         11         25      52         13
                 Subj07          7         68      15          9
                 Subj08          4         21      65         10
                 Subj09          7         18      66          8
                 Subj10         10         19      59         13
                 Subj12          9         52      24         15

                              Frozen Diet Entrees
                              Average Importances

                              The MEANS Procedure

                           Variable             Mean
                           --------------------------
                           Ingredient      8.9069044
                           Fat            38.0953010
                           Price          39.0198700
                           Calories       13.9779245
                           --------------------------
```

On the average, price is the most important attribute followed very closely by fat content. These two attributes on the average account for 77% of preference. Calories and main ingredient account for the remaining 23%. Note that everyone does not have the same pattern of importance values. However, it is a little hard to compare subjects just by looking at the numbers.

We can make a nicer display of importances with stars flagging the most important attributes for each product as follows:

```
data im2;
   set im;
   label c1 = 'Ingredient' c2 = 'Fat' c3 = 'Price' c4 = 'Calories';
   c1 = put(ingredient, 2.) || substr('  ******', 1, ceil(ingredient / 15));
   c2 = put(fat       , 2.) || substr('  ******', 1, ceil(fat        / 15));
   c3 = put(price     , 2.) || substr('  ******', 1, ceil(price      / 15));
   c4 = put(calories  , 2.) || substr('  ******', 1, ceil(calories   / 15));
   run;
```

```
proc print label;
   title2 'Importances';
   var c1-c4;
   id _depvar_;
   run;
```

These steps replace each importance variable with its formatted value followed by zero stars for 0 - 30, one star for 30 - 45, two stars for 45 - 60, three stars for 60 - 75, and so on. The value returned by the `ceil` function is the number of characters that are extracted from the string `' ******'`. The results are as follows:

```
                        Frozen Diet Entrees
                            Importances

     Subject    Ingredient    Fat          Price        Calories

     Subj01         13        50  **       24            13
     Subj02          8        65  ***      15            11
     Subj03          7        62  ***      21            11
     Subj04         13        22           25            39  *
     Subj05          7        17           64  ***       12
     Subj06         11        25           52  **        13
     Subj07          7        68  ***      15             9
     Subj08          4        21           65  ***       10
     Subj09          7        18           66  ***        8
     Subj10         10        19           59  **        13
     Subj12          9        52  **       24            15
```

Subject 4 is more concerned about calories. However, most individuals seem to fall into one of two groups, either primarily price conscious then fat conscious, or primarily fat conscious then price conscious.

Both the `out=` data set and the `outtest=` data set contain the part-worth utilities. In the `out=` data set, they are contained in the observations whose `_type_` value is `'M COEFFI'`. The part-worth utilities are the multiple regression coefficients. The names of the variables that contain the part-worth utilities are stored in the macro variable `&_trgind`, which is automatically created by PROC TRANSREG. The following step displays the part-worth utilities:

```
proc print data=results2 label;
   title2 'Part-Worth Utilities';
   where _type_ = 'M COEFFI';
   id _name_;
   var &_trgind;
   run;
```

The results are as follows:

```
                                Frozen Diet Entrees
                                Part-Worth Utilities

           Ingredient,    Ingredient,    Ingredient,     Fat, 8      Fat, 5
 _NAME_      Chicken          Beef          Turkey        Grams       Grams

 Subj01      1.55556       -2.11111        0.55556      -6.94444    -0.11111
 Subj02     -1.05556        0.11111        0.94444      -7.72222     0.11111
 Subj03      0.66667       -1.00000        0.33333      -7.66667    -0.16667
 Subj04     -2.11111        0.72222        1.38889      -2.77778    -0.27778
 Subj05      0.55556        0.55556       -1.11111      -1.77778    -0.27778
 Subj06      1.11111        0.61111       -1.72222      -2.88889    -0.55556
 Subj07      0.22222        0.72222       -0.94444      -7.61111    -0.27778
 Subj08      0.50000       -0.50000        0.00000      -2.33333     0.00000
 Subj09      0.61111       -1.05556        0.44444      -2.05556    -0.05556
 Subj10     -1.38889        0.94444        0.44444      -2.05556    -0.38889
 Subj12      0.83333        0.66667       -1.50000      -6.16667    -1.16667

            Fat, 2      Price,      Price,      Price,     Calories,    Calories,
 _NAME_     Grams       $2.59       $2.29      $1.99         350          250

 Subj01    7.05556    -3.44444     0.22222    3.22222     -1.83333      1.83333
 Subj02    7.61111    -1.88889     0.11111    1.77778     -1.33333      1.33333
 Subj03    7.83333    -2.66667     0.16667    2.50000     -1.33333      1.33333
 Subj04    3.05556    -3.44444     0.38889    3.05556     -5.05556      5.05556
 Subj05    2.05556    -7.27778     0.22222    7.05556     -1.33333      1.33333
 Subj06    3.44444    -6.22222    -0.88889    7.11111     -1.61111      1.61111
 Subj07    7.88889    -1.94444     0.38889    1.55556     -1.00000      1.00000
 Subj08    2.33333    -7.33333    -0.00000    7.33333     -1.16667      1.16667
 Subj09    2.11111    -7.38889    -0.05556    7.44444     -0.94444      0.94444
 Subj10    2.44444    -7.22222     0.27778    6.94444     -1.50000      1.50000
 Subj12    7.33333    -2.83333    -0.50000    3.33333     -2.00000      2.00000
```

These part-worth utilities can be clustered, for example, using PROC FASTCLUS, as follows:

```
proc fastclus data=results2 maxclusters=3 out=clusts;
   where _type_ = 'M COEFFI';
   id _name_;
   var &_trgind;
   run;

proc sort; by cluster; run;

proc print label;
   title2 'Part-Worth Utilities, Clustered';
   by cluster;
   id _name_;
   var &_trgind;
   run;
```

The results are as follows:

```
                              Frozen Diet Entrees
                         Part-Worth Utilities, Clustered

 ---------------------------------- Cluster=1 -----------------------------------

           Ingredient,    Ingredient,    Ingredient,     Fat, 8      Fat, 5
 _NAME_      Chicken          Beef          Turkey        Grams       Grams

 Subj05      0.55556        0.55556       -1.11111     -1.77778    -0.27778
 Subj06      1.11111        0.61111       -1.72222     -2.88889    -0.55556
 Subj08      0.50000       -0.50000        0.00000     -2.33333     0.00000
 Subj09      0.61111       -1.05556        0.44444     -2.05556    -0.05556
 Subj10     -1.38889        0.94444        0.44444     -2.05556    -0.38889

            Fat, 2     Price,       Price,       Price,     Calories,    Calories,
 _NAME_     Grams       $2.59        $2.29       $1.99        350          250

 Subj05    2.05556    -7.27778      0.22222     7.05556     -1.33333     1.33333
 Subj06    3.44444    -6.22222     -0.88889     7.11111     -1.61111     1.61111
 Subj08    2.33333    -7.33333     -0.00000     7.33333     -1.16667     1.16667
 Subj09    2.11111    -7.38889     -0.05556     7.44444     -0.94444     0.94444
 Subj10    2.44444    -7.22222      0.27778     6.94444     -1.50000     1.50000

 ---------------------------------- Cluster=2 -----------------------------------

           Ingredient,    Ingredient,    Ingredient,     Fat, 8      Fat, 5
 _NAME_      Chicken          Beef          Turkey        Grams       Grams

 Subj01      1.55556       -2.11111        0.55556     -6.94444    -0.11111
 Subj02     -1.05556        0.11111        0.94444     -7.72222     0.11111
 Subj03      0.66667       -1.00000        0.33333     -7.66667    -0.16667
 Subj07      0.22222        0.72222       -0.94444     -7.61111    -0.27778
 Subj12      0.83333        0.66667       -1.50000     -6.16667    -1.16667

            Fat, 2     Price,       Price,       Price,     Calories,    Calories,
 _NAME_     Grams       $2.59        $2.29       $1.99        350          250

 Subj01    7.05556    -3.44444      0.22222     3.22222     -1.83333     1.83333
 Subj02    7.61111    -1.88889      0.11111     1.77778     -1.33333     1.33333
 Subj03    7.83333    -2.66667      0.16667     2.50000     -1.33333     1.33333
 Subj07    7.88889    -1.94444      0.38889     1.55556     -1.00000     1.00000
 Subj12    7.33333    -2.83333     -0.50000     3.33333     -2.00000     2.00000
```

```
---------------------------------- Cluster=3 -----------------------------------

           Ingredient,    Ingredient,    Ingredient,     Fat, 8      Fat, 5
 _NAME_      Chicken          Beef          Turkey        Grams       Grams

 Subj04      -2.11111       0.72222        1.38889      -2.77778    -0.27778

            Fat, 2     Price,       Price,      Price,     Calories,    Calories,
 _NAME_     Grams       $2.59       $2.29       $1.99        350           250

 Subj04    3.05556    -3.44444     0.38889     3.05556     -5.05556      5.05556
```

The clusters reflect what we saw looking at the importance information. Subject 4, who is the only subject that is primarily calorie conscious, is in a separate cluster from everyone else. Cluster 1 subjects 5, 6, 8, 9, and 10 are primarily price conscious. Cluster 2 subjects 1, 2, 3, 7, and 12 are primarily fat conscious.

Spaghetti Sauce

This example uses conjoint analysis in a study of spaghetti sauce preferences. The goal is to investigate the main effects for all of the attributes and the interaction of brand and price, and to simulate market share. Rating scale data are gathered from a group of subjects. The example has eight parts.

- An efficient experimental design is generated with the `%MktEx` macro.
- Descriptions of the spaghetti sauces are generated.
- Data are collected, entered, and processed.
- The metric conjoint analysis is performed with PROC TRANSREG.
- Market share is simulated with the maximum utility model.
- Market share is simulated with the Bradley-Terry-Luce and logit models.
- The simulators are compared.
- Change in market share is investigated.

Create an Efficient Experimental Design with the %MktEx Macro

In this example, subjects were asked to rate their interest in purchasing hypothetical spaghetti sauces. The table shows the attributes, the attribute levels, and the number of *df* associated with each effect.

Experimental Design		
Effects	**Levels**	*df*
Intercept		1
Brand	Pregu, Sundance, Tomato Garden	2
Meat Content	Vegetarian, Meat, Italian Sausage	2
Mushroom Content	Mushrooms, No Mention	1
Natural Ingredients	All Natural Ingredients, No Mention	1
Price	\$1.99, \$2.29, \$2.49, \$2.79, \$2.99	4
Brand × Price		8

The brand names "Pregu", "Sundance", and "Tomato Garden" are artificial. Usually, real brand names would be used—your client's or company's brand and the competitors' brands. The absence of a feature (for example, no mushrooms) is not mentioned in the product description, hence the "No Mention" in the table.

In this design there are 19 model *df*. A design with more than 19 runs must be generated if there are to be error *df*. A popular heuristic is to limit the design size to at most 30 runs. In this example, 30 runs allow us to have two observations in each of the 15 brand by price cells. Note however that when subjects are required to make that many judgments, there is the risk that the quality of the data will be poor. Caution should be used when generating designs with this many runs. We can use the `%MktRuns` macro to evaluate this and other design sizes. See page 803 for macro documentation and information about installing and using SAS autocall macros. We specify the number of levels of each factor as the argument as follows:

```
title 'Spaghetti Sauces';

%mktruns(3 3 2 2 5)
```

The results are as follows:

```
                                    Spaghetti Sauces

                                    Design Summary

                                Number of
                                Levels       Frequency

                                     2            2
                                     3            2
                                     5            1

                     Saturated      = 11
                     Full Factorial = 180

            Some Reasonable                          Cannot Be
               Design Sizes       Violations     Divided By

                        180 *              0
                         60                1     9
                         90                1     4
                        120                1     9
                         30                2     4  9
                        150                2     4  9
                         36                5     5 10 15
                         72                5     5 10 15
                        108                5     5 10 15
                        144                5     5 10 15
                         11 S             15     2  3  4  5  6  9 10 15

             * - 100% Efficient design can be made with the MktEx macro.
             S - Saturated Design - The smallest design that can be made.
                 Note that the saturated design is not one of the
                 recommended designs for this problem.  It is shown
                 to provide some context for the recommended sizes.
```

We see that 30 is a reasonable size, although it cannot be divided by $9 = 3 \times 3$ and $4 = 2 \times 2$, so perfect orthogonality is not possible. We would need a much larger size like 60 or 180 to do better. Note that this output states "Saturated=11" referring to a main-effects model. In this example, we are also interested in the brand by price interaction. We can run the `%MktRuns` macro again, this time specifying the interaction as follows:

```
%mktruns(3 3 2 2 5, interact=1*5)
```

The results are as follows:

```
                              Spaghetti Sauces

                              Design Summary

                          Number of
                          Levels       Frequency

                              2            2
                              3            2
                              5            1

                              Spaghetti Sauces

              Saturated      = 19
              Full Factorial = 180

              Some Reasonable                     Cannot Be
                 Design Sizes     Violations      Divided By

                 180 *              0
                  90                1      4
                  60                2      9 45
                 120                2      9 45
                  30                3      4  9 45
                 150                3      4  9 45
                  36                8      5 10 15 30 45
                  72                8      5 10 15 30 45
                 108                8      5 10 15 30 45
                 144                8      5 10 15 30 45
                  19 S             18      2  3  4  5  6  9 10 15 30 45

          * - 100% Efficient design can be made with the MktEx macro.
          S - Saturated Design - The smallest design that can be made.
              Note that the saturated design is not one of the
              recommended designs for this problem.  It is shown
              to provide some context for the recommended sizes.
```

Now the output states "Saturated=19", which includes the 8 *df* for the interaction. We see as before that 30 cannot be divided by $4 = 2 \times 2$. We also see that 30 cannot be divide by $45 = 3 \times 15$ so each level of meat content cannot appear equally often in each brand/price cell. Since we would need a much larger size to do better, we will use 30 runs.

The next steps create and evaluate the design. First, formats for each of the factors are created using

PROC FORMAT. The `%MktEx` macro is called to create the design. The factors `x1 = Brand` and `x2 = Meat` are designated as three-level factors, `x3 = Mushroom` and `x4 = Ingredients` as two-level factors, and `x5 = Price` as a five-level factor. The `interact=1*5` option specifies that the interaction between the first and fifth factors must be estimable (`x1` $\times$ `x5` which is brand by price), `n=30` specifies the number of runs, and `seed=289` specifies the random number seed. The `where` macro provides restrictions that eliminate unrealistic combinations. Specifically, products at the cheapest price, $1.99, with meat, and products with Italian Sausage with All Natural Ingredients are eliminated from consideration.

We impose restrictions with the `%MktEx` macro by writing a macro, with IML statements, that quantifies the badness of each run of the design. The variable `bad` is set to zero when everything is fine; `bad` is set to values larger than zero when the row of the design does not conform to the restrictions. When there are multiple restrictions, as there are here, the variable `bad` is set to the number of violations, so the macro can know when it is moving in the right direction as it changes the design. *This is important!* The restrictions macro must quantify badness in a functional way (that is, not a binary okay or not okay) so that the `%MktEx` macro can see which direction it needs to head to find the minimum. If the `%MktEx` macro considers a change to the design that makes the design closer to what you want, this needs to be reflected in the badness criterion, otherwise `%MktEx` is less inclined to actually make the change.

The first five statements in the restrictions macro reformulate the internal factor names `x1-x5` and internal factor levels (positive integers beginning with one) into more meaningful names and levels. Brand is `'P'` (Pregu) when `x1 = 1`, `'S'` (Sundance) when `x1 = 2`, and `'T'` (Tomato Garden) when `x1 = 3`. Similarly, `x2-x5` are mapped to `Meat -- Price`, each with more mnemonic levels. See page 475) for more information about formulating restrictions based on mnemonic names and levels. Our first restriction (contribution to the badness value) is `(meat = 'I' & natural = 'A')` and our second is `(price = 1.99 & (meat = 'M' | meat = 'I'))`, where `&` means `and` and `|` means `or`.[*] The restrictions correspond to `(Meat = 'Italian Sausage' & Ingredients = 'All Natural')` and `(Price = 1.99 & (Meat = 'Meat' | Meat = 'Italian Sausage'))`, and you could set up the restrictions macro to use these longer levels if you want. Each of these Boolean or logical expressions evaluates to 1 when the expression is true and 0 when it is false. The sum of the two restrictions is: 0 - no problem, 1 - one restriction violation, or 2 - two restriction violations.

The `%MktLab` macro assigns actual descriptive factor names instead of the default `x1-x5` and formats for the levels. The default input to the `%MktLab` macro is the data set `Randomized`, which is the randomized design created by the `%MktEx` macro.

The default output from the `%MktLab` macro is a data set called `Final`. We instead use the `out=` option to store the results in a permanent SAS data set. The `%MktEval` macro is used to display the frequencies for each level, the two-way frequencies, and the number of times each product occurs in the design (five-way frequencies). The following steps create and evaluate the design:

[*]In the restrictions macro, you must use the logical symbols `| & ^ ¬ > < >= <= = ^= ¬=` and *not* the logical words `OR AND NOT GT LT GE LE EQ NE`. Furthermore, when specifying a range of values, you must use the syntax `a` <= `b` & `b` <= `c` not `a` <= `b` <= *c*.

```
title 'Spaghetti Sauces';

proc format;
   value br 1='Pregu'       2='Sundance'  3='Tomato Garden';
   value me 1='Vegetarian'  2='Meat'      3='Italian Sausage';
   value mu 1='Mushrooms'   2='No Mention';
   value in 1='All Natural' 2='No Mention';
   value pr 1='1.99' 2='2.29' 3='2.49' 4='2.79' 5='2.99';
   run;

%macro resmac;
   Brand    = {'P' 'S' 'T'}[x1];
   Meat     = {'V' 'M' 'I'}[x2];
   Mushroom = {'M' ' '}[x3];
   Natural  = {'A' ' '}[x4];
   Price    = {1.99 2.29 2.49 2.79 2.99}[x5];
   bad      = (meat = 'I' & natural = 'A') +
              (price = 1.99 & (meat = 'M' | meat = 'I'));
   %mend;

%mktex(3 3 2 2 5,                  /* all of the factor levels             */
       interact=1*5,               /* x1*x5 interaction                    */
       n=30,                       /* 30 runs                              */
       seed=289,                   /* random number seed                   */
       restrictions=resmac)        /* name of restrictions macro           */

%mktlab(data=randomized, vars=Brand Meat Mushroom Ingredients Price,
        statements=format brand br. meat me. mushroom mu.
                   ingredients in. price pr.,
        out=sasuser.spag)

%mkteval(data=sasuser.spag)

proc print data=sasuser.spag; run;
```

Some of the output from the %MktEx macro is as follows:

```
                             Spaghetti Sauces

                         Algorithm Search History

                             Current          Best
Design    Row,Col  D-Efficiency  D-Efficiency  Notes
----------------------------------------------------------
     1      Start       92.6280                Can
     1      2   1       92.6280       92.6280  Conforms
     1        End       92.6280

     2      Start       78.9640                Tab,Unb
     2     28   1       91.5726                Conforms
     2        End       91.6084

     3      Start       78.9640                Tab,Unb
     3      1   1       91.5434                Conforms
     3        End       91.6084

     4      Start       77.5906                Tab,Ran
     4     28   1       91.9486                Conforms
     4      5   4       92.6280       92.6280
     4        End       92.6280
     .
     .
     .
    21      Start       74.7430                Ran,Mut,Ann
    21     24   1       89.9706                Conforms
    21        End       91.6084
                             Spaghetti Sauces

                          Design Search History

                             Current          Best
Design    Row,Col  D-Efficiency  D-Efficiency  Notes
----------------------------------------------------------
     0    Initial       92.6280       92.6280  Ini

     1      Start       92.6280                Can
     1      2   1       92.6280       92.6280  Conforms
     1        End       92.6280
```

```
                              Spaghetti Sauces

                          Design Refinement History

                                  Current         Best
              Design    Row,Col  D-Efficiency  D-Efficiency  Notes
              ----------------------------------------------------------
                   0    Initial       92.6280       92.6280  Ini

                   1      Start       90.4842                Pre,Mut,Ann
                   1      2   1       91.2145                Conforms
                   1        End       91.6084
                   .
                   .
                   .

                   6      Start       91.1998                Pre,Mut,Ann
                   6      2   1       91.6084                Conforms
                   6        End       91.6084

   NOTE: Stopping since it appears that no improvement is possible.

                              Spaghetti Sauces

                            The OPTEX Procedure

                          Class Level Information

                         Class  Levels    -Values--

                            x1       3    1 2 3
                            x2       3    1 2 3
                            x3       2    1 2
                            x4       2    1 2
                            x5       5    1 2 3 4 5

                              Spaghetti Sauces

                            The OPTEX Procedure

                                                                    Average
                                                                   Prediction
          Design                                                    Standard
          Number     D-Efficiency     A-Efficiency     G-Efficiency       Error
          ------------------------------------------------------------------------
               1          92.6280          82.6056          97.6092          0.7958
```

The *D*-Efficiency looks reasonable at 92.63. For this problem, the full-factorial design is small (180 runs), and the macro found the same *D*-efficiency several times. This suggests that we have probably indeed found the optimal design for this situation. The results from the `%MktEval` macro are as follows:

```
                             Spaghetti Sauces
                  Canonical Correlations Between the Factors
              There are 2 Canonical Correlations Greater Than 0.316

                     Brand    Meat    Mushroom    Ingredients    Price

      Brand          1        0.21     0             0.17        0
      Meat           0.21     1        0.08          0.42        0.52
      Mushroom       0        0.08     1             0           0
      Ingredients    0.17     0.42     0             1           0.17
      Price          0        0.52     0             0.17        1

                             Spaghetti Sauces
                Canonical Correlations > 0.316 Between the Factors
              There are 2 Canonical Correlations Greater Than 0.316

                                          r    r Square

                  Meat    Price          0.52      0.27
                  Meat    Ingredients    0.42      0.17

                             Spaghetti Sauces
                          Summary of Frequencies
              There are 2 Canonical Correlations Greater Than 0.316
                        * - Indicates Unequal Frequencies

                                 Frequencies

          Brand                  10 10 10
     *    Meat                   15 9 6
          Mushroom               15 15
     *    Ingredients            12 18
          Price                  6 6 6 6 6
     *    Brand Meat             4 3 3 5 4 1 6 2 2
          Brand Mushroom         5 5 5 5 5 5
     *    Brand Ingredients      3 7 5 5 4 6
          Brand Price            2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
     *    Meat Mushroom          7 8 5 4 3 3
     *    Meat Ingredients       7 8 5 4 0 6
     *    Meat Price             6 3 2 2 2 0 2 2 3 2 0 1 2 1 2
     *    Mushroom Ingredients   6 9 6 9
          Mushroom Price         3 3 3 3 3 3 3 3 3 3
     *    Ingredients Price      3 3 2 2 2 3 3 4 4 4
          N-Way                  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                                 1 1 1 1 1 1 1 1 1 1 1
```

The meat and price factors are correlated, as are the meat and ingredients factors. This is not surprising since we excluded cells for these factor combinations and hence forced some correlations. The rest of the correlations are small.

The frequencies look good. The n-way frequencies at the end of this listing show that each product occurs only once, so there are no duplicates. Each brand, price, and brand/price combination occurs equally often, as does each mushroom level. There are more vegetarian sauces (the first formatted level) than either of the meat sauces because of the restrictions that meat cannot occur at the lowest price and Italian sausage cannot be paired with all-natural ingredients. The design is as follows:

```
                              Spaghetti Sauces

  Obs    Brand            Meat                Mushroom     Ingredients    Price

    1    Pregu            Meat               No Mention    No Mention     2.79
    2    Tomato Garden    Vegetarian         No Mention    No Mention     2.79
    3    Pregu            Meat               Mushrooms     All Natural    2.29
    4    Tomato Garden    Vegetarian         Mushrooms     All Natural    2.49
    5    Sundance         Vegetarian         Mushrooms     No Mention     1.99
    6    Pregu            Italian Sausage    No Mention    No Mention     2.49
    7    Tomato Garden    Vegetarian         No Mention    No Mention     2.99
    8    Tomato Garden    Italian Sausage    Mushrooms     No Mention     2.29
    9    Pregu            Vegetarian         Mushrooms     No Mention     2.49
   10    Pregu            Vegetarian         No Mention    No Mention     2.29
   11    Sundance         Vegetarian         Mushrooms     No Mention     2.79
   12    Tomato Garden    Vegetarian         Mushrooms     No Mention     1.99
   13    Sundance         Meat               No Mention    No Mention     2.29
   14    Sundance         Meat               Mushrooms     No Mention     2.99
   15    Pregu            Italian Sausage    Mushrooms     No Mention     2.79
   16    Tomato Garden    Italian Sausage    Mushrooms     No Mention     2.99
   17    Sundance         Vegetarian         Mushrooms     All Natural    2.29
   18    Pregu            Meat               Mushrooms     All Natural    2.99
   19    Tomato Garden    Meat               No Mention    No Mention     2.49
   20    Sundance         Meat               Mushrooms     All Natural    2.49
   21    Pregu            Vegetarian         No Mention    All Natural    1.99
   22    Sundance         Meat               No Mention    All Natural    2.79
   23    Tomato Garden    Vegetarian         No Mention    All Natural    1.99
   24    Sundance         Italian Sausage    No Mention    No Mention     2.49
   25    Sundance         Vegetarian         No Mention    All Natural    1.99
   26    Sundance         Vegetarian         No Mention    All Natural    2.99
   27    Pregu            Italian Sausage    No Mention    No Mention     2.99
   28    Tomato Garden    Vegetarian         No Mention    All Natural    2.29
   29    Pregu            Vegetarian         Mushrooms     No Mention     1.99
   30    Tomato Garden    Meat               Mushrooms     All Natural    2.79
```

Generating the Questionnaire

Next, preparations are made for data collection. A DATA step is used to print descriptions of each product combination, for example, as follows:

```
Try Pregu brand vegetarian spaghetti sauce, now with
mushrooms.  A 26 ounce jar serves four adults for only
$1.99.
```

Remember that "No Mention" is not mentioned. The following step prints the questionnaires including a cover sheet:

```
options ls=80 ps=74 nonumber nodate;
title;

data _null_;
   set sasuser.spag;
   length lines $ 500 aline $ 60;
   file print linesleft=ll;

   * Format meat level, preserve 'Italian' capitalization;
   aline = lowcase(put(meat, me.));
   if aline =: 'ita' then substr(aline, 1, 1) = 'I';

   * Format meat differently for 'vegetarian';
   if meat > 1
      then lines = 'Try ' || trim(put(brand, br.)) ||
                   ' brand spaghetti sauce with ' || aline;
      else lines = 'Try ' || trim(put(brand, br.)) ||
                   ' brand ' || trim(aline) || ' spaghetti sauce ';

   * Add mushrooms, natural ingredients to text line;
   n = (put(ingredients, in.) =: 'All');
   m = (put(mushroom,    mu.) =: 'Mus');

   if n or m then do;
      lines = trim(lines) || ', now with';

      if m then do;
         lines = trim(lines) || ' ' || lowcase(put(mushroom, mu.));
         if n then lines = trim(lines) || ' and';
         end;
      if n then lines = trim(lines) || ' ' ||
                        lowcase(put(ingredients, in.)) || ' ingredients';
      end;

   * Add price;
   lines = trim(lines) ||
          '.  A 26 ounce jar serves four adults for only $' ||
          put(price, pr.) || '.';
   * Print cover page, with subject number, instructions, and rating scale;
   if _n_ = 1 then do;
      put ///// +41 'Subject: ________' ////
          +5 'Please rate your willingness to purchase the following' /
          +5 'products on a nine point scale.' ///
          +9 '1   Definitely Would Not Purchase This Product' ///
          +9 '2'  ///
          +9 '3   Probably Would Not Purchase This Product' ///
          +9 '4'  ///
          +9 '5   May or May Not Purchase This Product' ///
          +9 '6'  ///
          +9 '7   Probably Would Purchase This Product' ///
          +9 '8'  ///
          +9 '9   Definitely Would Purchase This Product' /////
          +5 'Please rate every product and be sure to rate' /
          +5 'each product only once.' //////
          +5 'Thank you for your participation!';
      put _page_;
      end;

   if ll < 8 then put _page_;

   * Break up description, print on several lines;

   start = 1;
   do l = 1 to 10 until(aline = ' ');

      * Find a good place to split, blank or punctuation;
      stop = start + 60;
      do i = stop to start by -1 while(substr(lines, i, 1) ne ' '); end;
      do j = i to max(start, i - 8) by -1;
         if substr(lines, j, 1) in ('.' ',') then do; i = j; j = 0; end;
         end;

      stop = i; len = stop + 1 - start;
      aline = substr(lines, start, len);
      start = stop + 1;
      if l = 1 then put +5 _n_ 2. ') ' aline;
      else          put +9 aline;
      end;

   * Print rating scale;
   put +9 'Definitely                                      Definitely ' /
       +9 'Would Not   1   2   3   4   5   6   7   8   9   Would      ' /
       +9 'Purchase                                        Purchase   ' //;
   run;

options ls=80 ps=60 nonumber nodate;
```

Some of the results are as follows:

```
                                        Subject: ________



        Please rate your willingness to purchase the following
        products on a nine point scale.


            1   Definitely Would Not Purchase This Product


            2


            3   Probably Would Not Purchase This Product


            4


            5   May or May Not Purchase This Product


            6


            7   Probably Would Purchase This Product


            8


            9   Definitely Would Purchase This Product




        Please rate every product and be sure to rate
        each product only once.




        Thank you for your participation!
```

1) Try Pregu brand spaghetti sauce with meat. A 26 ounce jar serves four adults for only $2.79.

Definitely										Definitely
Would Not	1	2	3	4	5	6	7	8	9	Would
Purchase										Purchase

2) Try Tomato Garden brand vegetarian spaghetti sauce. A 26 ounce jar serves four adults for only $2.79.

Definitely										Definitely
Would Not	1	2	3	4	5	6	7	8	9	Would
Purchase										Purchase

3) Try Pregu brand spaghetti sauce with meat, now with mushrooms and all natural ingredients. A 26 ounce jar serves four adults for only $2.29.

Definitely										Definitely
Would Not	1	2	3	4	5	6	7	8	9	Would
Purchase										Purchase

4) Try Tomato Garden brand vegetarian spaghetti sauce, now with mushrooms and all natural ingredients. A 26 ounce jar serves four adults for only $2.49.

Definitely										Definitely
Would Not	1	2	3	4	5	6	7	8	9	Would
Purchase										Purchase

5) Try Sundance brand vegetarian spaghetti sauce, now with mushrooms. A 26 ounce jar serves four adults for only $1.99.

Definitely										Definitely
Would Not	1	2	3	4	5	6	7	8	9	Would
Purchase										Purchase

6) Try Pregu brand spaghetti sauce with Italian sausage. A 26 ounce jar serves four adults for only $2.49.

Definitely										Definitely
Would Not	1	2	3	4	5	6	7	8	9	Would
Purchase										Purchase

7) Try Tomato Garden brand vegetarian spaghetti sauce. A 26 ounce jar serves four adults for only $2.99.

Definitely										Definitely
Would Not	1	2	3	4	5	6	7	8	9	Would
Purchase										Purchase

```
      .
      .
      .

    30) Try Tomato Garden brand spaghetti sauce with meat, now with
        mushrooms and all natural ingredients.  A 26 ounce jar
        serves four adults for only $2.79.

        Definitely                                      Definitely
        Would Not   1   2   3   4   5   6   7   8   9   Would
        Purchase                                        Purchase
```

In the interest of space, not all questions are printed.

Data Processing

The following DATA step reads the input data:

```
title 'Spaghetti Sauces';

data rawdata;
   missing _;
   input subj @5 (rate1-rate30) (1.);
   name = compress('Sub' || put(subj, z3.));
   if nmiss(of rate:) = 0;
   datalines;
  1 319591129691132168146121171191
  2 749173216928911175549891841791
  3 449491116819413186158171961791
  .
  .
  .
 14 1139812951994_9466149198915699
  .
  .
  .
 19 2214922399981121.1116161941991
  .
  .
  .
;
```

Only a portion of the input data set is displayed. Some cases have ordinary '.' missing values. This code was used at data entry for no response. When there were multiple responses or the response was not clear, the special underscore missing value was used. The statement `missing _` specifies that underscore missing values are to be expected in the data. The `input` statement reads the subject number and the 30 ratings. A name like `Subj001`, `Subj002`, ..., `Subj030` is created from the subject number. If there are any missing data, all data for that subject are excluded by the `if nmiss(of` `rate:) = 0` statement. Next, the data are transposed from one row per subject and 30 columns to one column per subject and 30 rows, one for each product rated. Then the data are merged with the experimental design. The following steps do this final processing:

```
proc transpose data=rawdata(drop=subj) out=temp(drop=_name_);
    id name;
    run;

data inputdata; merge sasuser.spag temp; run;
```

Metric Conjoint Analysis

Next, we use PROC TRANSREG to perform the conjoint analysis as follows:

```
ods exclude notes mvanova anova;
proc transreg data=inputdata utilities short separators='' ', '
   lprefix=0 outtest=utils method=morals;
   title2 'Conjoint Analysis';
   model identity(sub:) =
         class(brand | price meat mushroom ingredients / zero=sum);
   output p ireplace out=results1 coefficients;
   run;
```

The `utilities` option requests conjoint analysis output, and the `short` option suppresses the iteration histories. The `lprefix=0` option specifies that zero variable name characters are to be used to construct the labels for the part-worths; the labels simply consist of formatted values. The `outtest=` option creates an output SAS data set, `Utils`, that contains all of the statistical results. The `method=morals`, algorithm fits the conjoint analysis model separately for each subject. We specify `ods exclude notes mvanova anova` to exclude ANOVA information (which we usually want to ignore) and provide more parsimonious output.

The `model` statement names the ratings for each subject as dependent variables and the factors as independent variables. Since this is a metric conjoint analysis, `identity` is specified for the ratings. The `identity` transformation is the no-transformation option, which is used for variables that need to enter the model with no further manipulations. The factors are specified as `class` variables, and the `zero=sum` option is specified to constrain the parameter estimates to sum to zero within each effect. The `brand | price` specification asks for a simple `brand` effect, a simple `price` effect, and the `brand * price` interaction.

The `p` option in the `output` statement requests predicted values, the `ireplace` option suppresses the output of transformed independent variables, and the `coefficients` option outputs the part-worth utilities. These options control the contents of the `out=results` data set, which contains the ratings, predicted utilities for each product, indicator variables, and the part-worth utilities.

In the interest of space, only the results for the first subject are displayed here. Recall that we used an `ods exclude` statement and we used PROC TEMPLATE on page 683 to customize the output from PROC TRANSREG. The results are as follows:

```
                             Conjoint Analysis

                           The TRANSREG Procedure

                          Class Level Information

            Class          Levels    Values

            Brand               3    Pregu  Sundance  Tomato Garden

            Price               5    1.99  2.29  2.49  2.79  2.99

            Meat                3    Vegetarian  Meat  Italian Sausage

            Mushroom            2    Mushrooms  No Mention

            Ingredients         2    All Natural  No Mention

                 Number of Observations Read            30
                 Number of Observations Used            30

                             Conjoint Analysis

                           The TRANSREG Procedure

      Identity(Sub001)
      Algorithm converged.

             The TRANSREG Procedure Hypothesis Tests for Identity(Sub001)


                   Root MSE           2.09608    R-Square    0.8344
                   Dependent Mean     3.73333    Adj R-Sq    0.5635
                   Coeff Var         56.14499

                            Part-Worth Utilities

                                                         Importance
                                              Standard   (% Utility
               Label                 Utility     Error       Range)

               Intercept              3.0675   0.45364

               Pregu                  2.0903   0.55937       28.924
               Sundance               0.2973   0.55886
               Tomato Garden         -2.3876   0.55205
```

```
1.99                 -0.6836     0.91331         7.134
2.29                  0.3815     0.77035
2.49                  0.4209     0.78975
2.79                 -0.5397     0.79677
2.99                  0.4209     0.78975

Pregu, 1.99           0.7430     1.09161        15.639
Pregu, 2.29           0.9491     1.13055
Pregu, 2.49          -0.7433     1.14528
Pregu, 2.79          -1.0115     1.13157
Pregu, 2.99           0.0626     1.13769
Sundance, 1.99        0.0361     1.09135
Sundance, 2.29       -1.2578     1.09310
Sundance, 2.49       -0.1443     1.16287
Sundance, 2.79        1.1633     1.09574
Sundance, 2.99        0.2027     1.12077
Tomato Garden, 1.99  -0.7791     1.08788
Tomato Garden, 2.29   0.3087     1.16798
Tomato Garden, 2.49   0.8876     1.16026
Tomato Garden, 2.79  -0.1518     1.10376
Tomato Garden, 2.99  -0.2654     1.13455

Vegetarian            2.2828     0.68783        27.813
Meat                 -0.2596     0.70138
Italian Sausage      -2.0231     0.86266

Mushrooms             1.5514     0.38441        20.042
No Mention           -1.5514     0.38441

All Natural          -0.0347     0.45814         0.448
No Mention            0.0347     0.45814
```

The next steps process the `outtest=` data set, saving the R square, adjusted R square, and *df*. Subjects whose adjusted R square is less than 0.3 (R square approximately 0.73) are flagged for exclusion. We want the final analysis to be based on subjects who seemed to be taking the task seriously. The following steps flag the subjects whose fit seems bad and create a macro variable `&droplist` that contains a list of variables to be dropped from the final analysis:

```
data model;
   set utils;
   if statistic in ('R-Square', 'Adj R-Sq', 'Model');
   Subj = scan(_depvar_, 2);
   if statistic = 'Model' then do;
      value = numdf;
      statistic = 'Num DF';
      output;
      value = dendf;
      statistic = 'Den DF';
      output;
      value = dendf + numdf + 1;
      statistic = 'N';
      end;
   output;
   keep statistic value subj;
   run;

proc transpose data=model out=summ;
   by subj;
   idlabel statistic;
   id statistic;
   run;

data summ2(drop=list);
   length list $ 1000;
   retain list;
   set summ end=eof;
   if adj_r_sq < 0.3 then do;
      Small = '*';
      list = trim(list) || ' ' || subj;
      end;
   if eof then call symput('droplist', trim(list));
   run;

%put &droplist;

proc print label data=summ2(drop=_name_ _label_); run;
```

The `outtest=` data set contains for each subject the ANOVA, R square, and part-worth utility tables. The numerator *df* is found in the variable `NumDF`, the denominator *df* is found in the variable `DenDF`, and the R square, and adjusted R square are found in the variable `Value`. The first DATA step processes the `outtest=` data set, stores all of the statistics of interest in the variable `Value`, and discards the extra observations and variables. The PROC TRANSPOSE step creates a data set with one observation per subject. The `&droplist` macro variable is as follows:

```
Sub011 Sub021 Sub031 Sub051 Sub071 Sub081 Sub092 Sub093 Sub094 Sub096
```

Some of the R square and *df* summary follows:

```
                                 Conjoint Analysis

                          Num     Den                           Adj
         Obs      Subj     DF      DF       N     R-Square       R-Sq      Small

           1     Sub001     18      11      30      0.83441    0.56345
           2     Sub002     18      11      30      0.91844    0.78497
           3     Sub003     18      11      30      0.92908    0.81302
           .
           .
           .
          10     Sub010     18      11      30      0.97643    0.93786
           .
           .
           .
          84     Sub091     18      11      30      0.85048     0.60581
          85     Sub092     18      11      30      0.64600     0.06673        *
          86     Sub093     18      11      30      0.45024    -0.44936        *
          87     Sub094     18      11      30      0.62250     0.00477        *
          88     Sub095     18      11      30      0.85996     0.63081
          89     Sub096     18      11      30      0.73321     0.29664        *
          90     Sub097     18      11      30      0.94155     0.84589
          91     Sub099     18      11      30      0.88920     0.70789
          92     Sub100     18      11      30      0.90330     0.74507
```

We see the *df* are right, and most of the R squares look good.

We can run the conjoint again, this time using the `drop=&droplist` data set option to drop the subjects with poor fit. In the interest of space, the `noprint` option is specified on this step. The output is the same as in the previous step, except for the fact that a few subject's tables are deleted. The following step performs the analysis:

```
proc transreg data=inputdata(drop=&droplist) utilities short noprint
   separators=’’ ’, ’ lprefix=0 outtest=utils method=morals;
   title2 ’Conjoint Analysis’;
   model identity(sub:) =
         class(brand | price meat mushroom ingredients / zero=sum);
   output p ireplace out=results2 coefficients;
   run;
```

Simulating Market Share

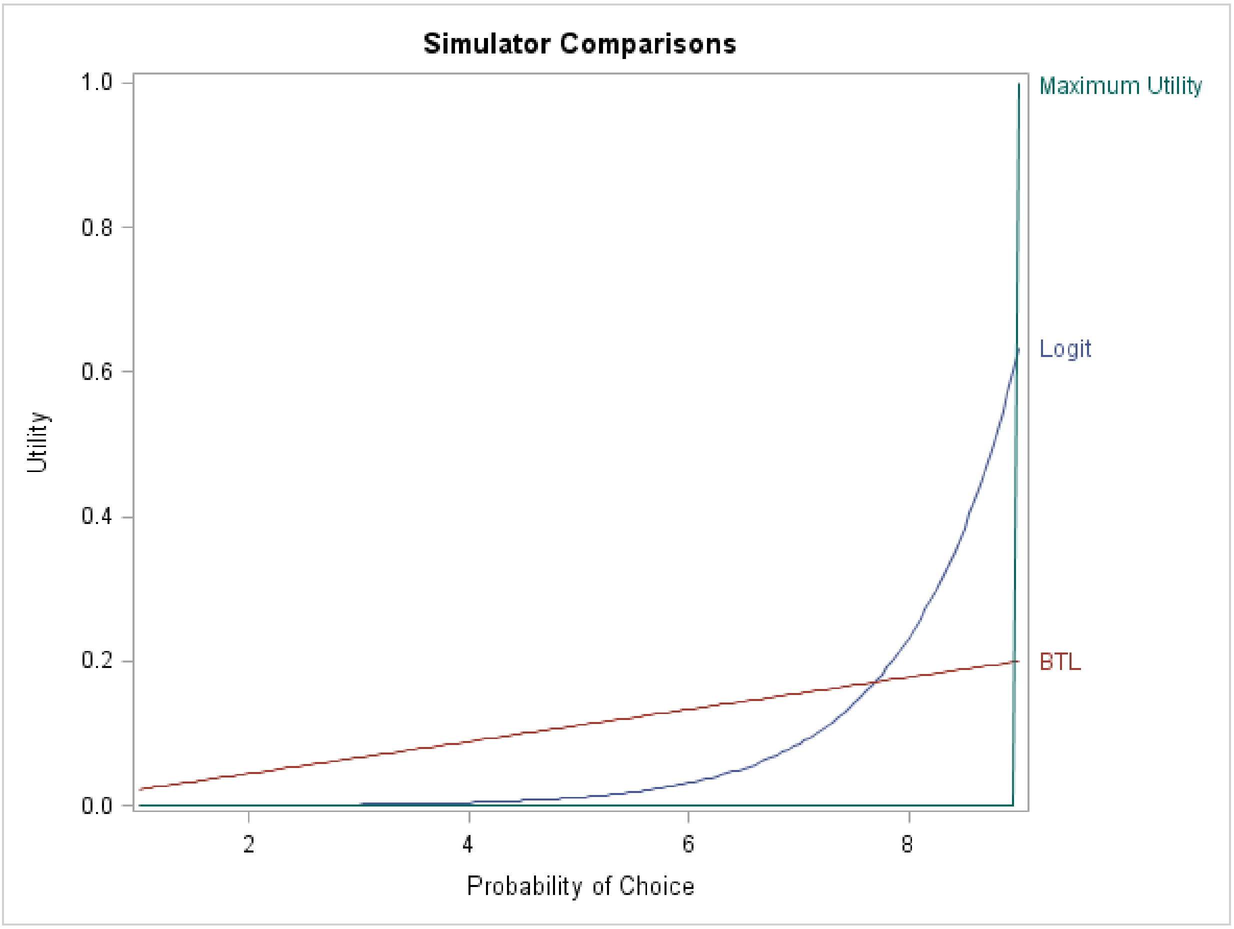
In many conjoint analysis studies, the conjoint analysis is not the primary goal. The conjoint analysis is used to generate part-worth utilities, which are then used as input to consumer choice and market share simulators. The end result for a product is its expected "preference share," which when properly weighted can be used to predict the proportion of times that the product will be purchased. The effects on market share of introducing new products can also be simulated.

One of the most popular ways to simulate market share is with the maximum utility model, which assumes each subject will buy with probability one the product for which he or she has the highest utility. The probabilities for each product are averaged across subjects to get predicted market share.

Other simulation methods include the Bradley-Terry-Luce (BTL) model and the logit model. Unlike the maximum utility model, the BTL and the logit models do not assign all of the probability of choice to the most preferred alternative. Probability is a continuous function of predicted utility. In the maximum utility model, probability of choice is a binary step function of utility. In the BTL model, probability of choice is a linear function of predicted utility. In the logit model, probability of choice is an increasing nonlinear logit function of predicted utility. The BTL model computes the probabilities by dividing each utility by the sum of the predicted utilities within each subject. The logit model divides the exponentiated predicted utilities by the sum of exponentiated utilities, again within subject.

$$\begin{array}{llll}
\text{Maximum Utility:} & p_{ijk} & = & 1.0 \ \ \text{if} \ \ y_{ijk} = \text{MAX}(y_{ijk}), \\
 & & & 0.0 \ \ \text{otherwise} \\
 & & & \\
\text{BTL:} & p_{ijk} & = & y_{ijk} / \sum\sum\sum y_{ijk} \\
 & & & \\
\text{Logit:} & p_{ijk} & = & \exp(y_{ijk}) / \sum\sum\sum \exp(y_{ijk})
\end{array}$$

The following plot shows the different assumptions made by the three choice simulators. This plot shows expected market share for a subject with utilities ranging from one to nine.



The maximum utility line is flat at zero until it reaches the maximum utility, where it jumps to 1.0. The BTL line increases from 0.02 to 0.20 as utility ranges from 1 to 9. The logit function increases exponentially, with small utilities mapping to near-zero probabilities and the largest utility mapping to a proportion of 0.63.

The maximum utility, BTL, and logit models are based on different assumptions and produce different results. The maximum utility model has the advantage of being scale-free. Any strictly monotonic transformation of each subject's predicted utilities produces the same market share. However, this model is unstable because it assigns a zero probability of choice to all alternatives that do not have the maximum predicted utility, including those that have predicted utilities near the maximum. The disadvantage of the BTL and logit models is that results are not invariant under linear transformations of the predicted utilities. These methods are considered inappropriate by some researchers for this reason. With negative predicted utilities, the BTL method produces negative probabilities, which are invalid. The BTL results change when a constant is added to the predicted utilities but do not change when a constant is multiplied by the predicted utilities. Conversely, the logit results change when a constant is multiplied by the predicted utilities but do not change when a constant is added to the predicted utilities. The BTL method is not often used in practice, the logit model is sometimes used, and the maximum utility model is most often used. See Finkbeiner (1988) for a discussion of conjoint analysis choice simulators. Do not confuse a logit model choice simulator and the multinomial logit model; they are quite different.

The three simulation methods produce different results. This is because all three methods make different assumptions about how consumers translate utility into choice. To see why the models differ, imagine a product that is everyone's second choice. Further imagine that there is wide-spread disagreement on first choice. Every other product is someone's first choice, and all other products are preferred about equally often. In the maximum utility model, this second choice product has zero probability of choice because no one would choose it first. In the other models, it should be the most preferred, because for every individual it has a high, near-maximum probability of choice. Of course, preference patterns are not usually as weird as the one just described. If consumers are perfectly rational and always choose the alternative with the highest utility, then the maximum utility model is correct. However, you need to be aware that your results will depend on the choice of simulator model and in BTL and logit, the scaling of the utilities. One reason why the discrete choice model is popular in marketing research is discrete choice models choices directly, whereas conjoint simulates choices indirectly.

The following steps produce the plot:

```
%let min = 1;
%let max = 9;
%let by  = 1;
%let inter = 20;
%let list = &min to &max by &by;
data a;
   do u = &list;
      logit = exp(u);
      btl   = u;
      sumb  + btl;
      suml  + logit;
      end;
      do u = &list / &inter;
         logit = exp(u);
         btl = u;
         max = abs(u - (&max)) < (0.5 * (&by / &inter));
         btl = btl / sumb;
         logit = logit / suml;
         output;
         end;
      label u = 'Probability of Choice';
      run;

   proc sgplot data=a;
      title 'Simulator Comparisons';
      series x=u y=logit / curvelabel='Logit' lineattrs=graphdata1;
      series x=u y=btl   / curvelabel='BTL' lineattrs=graphdata2;
      series x=u y=max   / curvelabel='Maximum Utility' lineattrs=graphdata3;
      yaxis label='Utility';
      run;
```

You can try this program with different minima and maxima to see the effects of linear transformations of the predicted utilities.

Simulating Market Share, Maximum Utility Model

This section shows how to use the predicted utilities from a conjoint analysis to simulate choice and predict market share. The end result for a hypothetical product is its expected market share, which is a prediction of the proportion of times that the product will be purchased. Note however, that a term like "expected market share," while widely used, is a misnomer. Without purchase volume data, it is unlikely that these numbers would mirror true market share. Nevertheless, conjoint analysis is a useful and popular marketing research technique.

A SAS macro is used to simulate market share. It takes a `method=morals` output data set from PROC TRANSREG and creates a data set with expected market share for each combination. First, market share is computed with the maximum utility model. The macro finds the most preferred combination(s) for each subject, which are those combinations with the largest predicted utility, and assigns the probability that each combination will be purchased. Typically, with the maximum utility model, one product for each subject has a probability of purchase of 1.0, and all other products have zero probability of purchase. However, when two predicted utilities both equal the maximum, that subject has two probabilities of 0.5, and the rest are zero. The probabilities are averaged across subjects for each product to get market share. Subjects can be differentially weighted. The following steps define and invoke the macro:

```
                           /*---------------------------------------*/
                           /* Simulate Market Share                 */
                           /*---------------------------------------*/
%macro sim(data=_last_, /* SAS data set with utilities.         */
           idvars=,     /* Additional variables to display with */
                        /* market share results.                */
           weights=,    /* By default, each subject contributes */
                        /* equally to the market share          */
                        /* computations.  To differentially     */
                        /* weight the subjects, specify a vector */
                        /* of weights, one per subject.         */
                        /* Separate the weights by blanks.      */
           out=shares,  /* Output data set name.                */
           method=max   /* max   - maximum utility model.       */
                        /* btl   - Bradley-Terry-Luce model.    */
                        /* logit - logit model.                 */
                        /* WARNING: The Bradley-Terry-Luce model */
                        /* and the logit model results are not   */
                        /* invariant under linear                */
                        /* transformations of the utilities.     */
          );            /*---------------------------------------*/

options nonotes;

%if &method = btl or &method = logit %then
   %put WARNING: The Bradley-Terry-Luce model and the logit model
results are not invariant under linear transformations of the
utilities.;
%else %if &method ne max %then %do;
   %put WARNING: Invalid method &method..  Assuming method=max.;
   %let method = max;
   %end;

* Eliminate coefficient observations, if any;
data temp1;
   set &data(where=(_type_ = 'SCORE' or _type_ = ' '));
   run;

* Determine number of runs and subjects.;
proc sql;
   create table temp2 as select nruns,
      count(nruns) as nsubs, count(distinct nruns) as chk
      from (select count(_depvar_) as nruns
      from temp1 where _type_ in ('SCORE', ' ') group by _depvar_);
   quit;
```

```
data _null_;
   set temp2;
   call symput('nruns', compress(put(nruns, 5.0)));
   call symput('nsubs', compress(put(nsubs, 5.0)));
   if chk > 1 then do;
      put 'ERROR: Corrupt input data set.';
      call symput('okay', 'no');
      end;
   else call symput('okay', 'yes');
   run;

%if &okay ne yes %then %do;
   proc print;
     title2 'Number of runs should be constant across subjects';
     run;
   %goto endit;
   %end;

%else %put NOTE: &nruns runs and &nsubs subjects.;

%let w = %scan(&weights, %eval(&nsubs + 1), %str( ));
%if %length(&w) > 0 %then %do;
   %put ERROR: Too many weights.;
   %goto endit;
   %end;

* Form nruns by nsubs data set of utilities;
data temp2;
   keep _u1 - _u&nsubs &idvars;
   array u[&nsubs] _u1 - _u&nsubs;

   do j = 1 to &nruns;

      * Read ID variables;
      set temp1(keep=&idvars) point = j;

      * Read utilities;
      k = j;
      do i = 1 to &nsubs;
         set temp1(keep=p_depend_) point = k;
         u[i] = p_depend_;
         %if &method = logit %then u[i] = exp(u[i]);;
         k = k + &nruns;
         end;

      output;
      end;

   stop;
   run;
```

```
* Set up for maximum utility model;
%if &method = max %then %do;

   * Compute maximum utility for each subject;
   proc means data=temp2 noprint;
      var _u1-_u&nsubs;
      output out=temp1 max=_sum1 - _sum&nsubs;
      run;

   * Flag maximum utility;
   data temp2(keep=_u1 - _u&nsubs &idvars);
      if _n_ = 1 then set temp1(drop=_type_ _freq_);
      array u[&nsubs] _u1 - _u&nsubs;
      array m[&nsubs] _sum1 - _sum&nsubs;
      set temp2;
      do i = 1 to &nsubs;
         u[i] = ((u[i] - m[i]) > -1e-8); /* < 1e-8 is considered 0 */
         end;
      run;

   %end;

* Compute sum for each subject;
proc means data=temp2 noprint;
   var _u1-_u&nsubs;
   output out=temp1 sum=_sum1 - _sum&nsubs;
   run;

* Compute expected market share;
data &out(keep=share &idvars);
   if _n_ = 1 then set temp1(drop=_type_ _freq_);
   array u[&nsubs] _u1 - _u&nsubs;
   array m[&nsubs] _sum1 - _sum&nsubs;
   set temp2;

   * Compute final probabilities;
   do i = 1 to &nsubs;
      u[i] = u[i] / m[i];
      end;

   * Compute expected market share;
   %if %length(&weights) = 0 %then %do;
      Share = mean(of _u1 - _u&nsubs);
      %end;
      %else %do;
         Share = 0;
         wsum  = 0;
         %do i = 1 %to &nsubs;
            %let w = %scan(&weights, &i, %str( ));
            %if %length(&w) = 0 %then %let w = .;
            if &w < 0 then do;
               if _n_ > 1 then stop;
               put "ERROR: Invalid weight &w..";
               call symput('okay', 'no');
               end;
            share = share + &w * _u&i;
            wsum  = wsum  + &w;
            %end;
         share = share / wsum;
         %end;
      run;

   options notes;

   %if &okay ne yes %then %goto endit;

   proc sort;
      by descending share &idvars;
      run;

   proc print label noobs;
      title2 'Expected Market Share';
      title3 %if       &method = max %then "Maximum Utility Model";
             %else %if &method = btl %then "Bradley-Terry-Luce Model";
             %else                         "Logit Model";;
      run;

   %endit:

   %mend;

   title 'Spaghetti Sauces';

   %sim(data=results2, out=maxutils, method=max,
        idvars=price brand meat mushroom ingredients);
```

The results are as follows:

```
                          Spaghetti Sauces
                       Expected Market Share
                       Maximum Utility Model

Brand           Price   Meat              Mushroom    Ingredients    Share

Sundance        1.99    Vegetarian        Mushrooms   No Mention    0.18293
Pregu           1.99    Vegetarian        No Mention  All Natural   0.14228
Tomato Garden   2.29    Italian Sausage   Mushrooms   No Mention    0.12195
Pregu           2.29    Vegetarian        No Mention  No Mention    0.10976
Pregu           1.99    Vegetarian        Mushrooms   No Mention    0.10366
Tomato Garden   1.99    Vegetarian        Mushrooms   No Mention    0.09146
Tomato Garden   1.99    Vegetarian        No Mention  All Natural   0.07520
Sundance        2.29    Vegetarian        Mushrooms   All Natural   0.07317
Sundance        1.99    Vegetarian        No Mention  All Natural   0.05081
Pregu           2.29    Meat              Mushrooms   All Natural   0.02439
Sundance        2.29    Meat              No Mention  No Mention    0.01220
Sundance        2.49    Italian Sausage   No Mention  No Mention    0.01220
Tomato Garden   2.29    Vegetarian        No Mention  All Natural   0.00000
Pregu           2.49    Vegetarian        Mushrooms   No Mention    0.00000
Pregu           2.49    Italian Sausage   No Mention  No Mention    0.00000
Sundance        2.49    Meat              Mushrooms   All Natural   0.00000
Tomato Garden   2.49    Vegetarian        Mushrooms   All Natural   0.00000
Tomato Garden   2.49    Meat              No Mention  No Mention    0.00000
Pregu           2.79    Meat              No Mention  No Mention    0.00000
Pregu           2.79    Italian Sausage   Mushrooms   No Mention    0.00000
Sundance        2.79    Vegetarian        Mushrooms   No Mention    0.00000
Sundance        2.79    Meat              No Mention  All Natural   0.00000
Tomato Garden   2.79    Vegetarian        No Mention  No Mention    0.00000
Tomato Garden   2.79    Meat              Mushrooms   All Natural   0.00000
Pregu           2.99    Meat              Mushrooms   All Natural   0.00000
Pregu           2.99    Italian Sausage   No Mention  No Mention    0.00000
Sundance        2.99    Vegetarian        No Mention  All Natural   0.00000
Sundance        2.99    Meat              Mushrooms   No Mention    0.00000
Tomato Garden   2.99    Vegetarian        No Mention  No Mention    0.00000
Tomato Garden   2.99    Italian Sausage   Mushrooms   No Mention    0.00000
```

The largest market share (18.29%) is for Sundance brand vegetarian sauce with mushrooms costing $1.99. The next largest share (14.23%) is Pregu brand vegetarian sauce with all natural ingredients costing $1.99. Five of the seven most preferred sauces all cost $1.99—the minimum. It is not clear from this simulation if any brand is the leader.

Simulating Market Share, Bradley-Terry-Luce and Logit Models

The Bradley-Terry-Luce model and the logit model are also available in the %SIM macro. These methods are illustrated in the following steps:

```
title 'Spaghetti Sauces';

%sim(data=results2, out=btl, method=btl,
     idvars=price brand meat mushroom ingredients);

%sim(data=results2, out=logit, method=logit,
     idvars=price brand meat mushroom ingredients);
```

The results are as follows:

```
                          Spaghetti Sauces
                       Expected Market Share
                     Bradley-Terry-Luce Model

Brand            Price   Meat               Mushroom     Ingredients     Share

Pregu            1.99    Vegetarian        Mushrooms    No Mention    0.053479
Sundance         1.99    Vegetarian        Mushrooms    No Mention    0.052990
Tomato Garden    1.99    Vegetarian        Mushrooms    No Mention    0.051751
Pregu            1.99    Vegetarian        No Mention   All Natural   0.050683
Sundance         1.99    Vegetarian        No Mention   All Natural   0.050193
Tomato Garden    1.99    Vegetarian        No Mention   All Natural   0.048955
Sundance         2.29    Vegetarian        Mushrooms    All Natural   0.048236
Pregu            2.29    Vegetarian        No Mention   No Mention    0.043972
Tomato Garden    2.29    Vegetarian        No Mention   All Natural   0.042035
Pregu            2.49    Vegetarian        Mushrooms    No Mention    0.041532
Pregu            2.29    Meat              Mushrooms    All Natural   0.041063
Sundance         2.29    Meat              No Mention   No Mention    0.036321
Tomato Garden    2.29    Italian Sausage   Mushrooms    No Mention    0.032995
Sundance         2.79    Vegetarian        Mushrooms    No Mention    0.032067
Sundance         2.49    Meat              Mushrooms    All Natural   0.031310
Tomato Garden    2.49    Vegetarian        Mushrooms    All Natural   0.031057
Sundance         2.99    Vegetarian        No Mention   All Natural   0.026879
Pregu            2.49    Italian Sausage   No Mention   No Mention    0.026046
Pregu            2.99    Meat              Mushrooms    All Natural   0.025318
Pregu            2.79    Meat              No Mention   No Mention    0.025038
Tomato Garden    2.79    Vegetarian        No Mention   No Mention    0.024325
Pregu            2.79    Italian Sausage   Mushrooms    No Mention    0.024263
Sundance         2.49    Italian Sausage   No Mention   No Mention    0.022383
Sundance         2.99    Meat              Mushrooms    No Mention    0.022264
Tomato Garden    2.99    Vegetarian        No Mention   No Mention    0.022113
Sundance         2.79    Meat              No Mention   All Natural   0.021858
Tomato Garden    2.79    Meat              Mushrooms    All Natural   0.021415
Tomato Garden    2.49    Meat              No Mention   No Mention    0.019142
Pregu            2.99    Italian Sausage   No Mention   No Mention    0.016391
Tomato Garden    2.99    Italian Sausage   Mushrooms    No Mention    0.013926

                          Spaghetti Sauces
                       Expected Market Share
                            Logit Model

 Brand            Price   Meat               Mushroom     Ingredients    Share

 Sundance         1.99    Vegetarian        Mushrooms    No Mention    0.10463
 Pregu            1.99    Vegetarian        No Mention   All Natural   0.09621
 Tomato Garden    1.99    Vegetarian        Mushrooms    No Mention    0.09001
 Pregu            1.99    Vegetarian        Mushrooms    No Mention    0.08358
```

```
    Pregu           2.29   Vegetarian        No Mention   No Mention    0.07755
    Sundance        2.29   Vegetarian        Mushrooms    All Natural   0.07102
    Tomato Garden   1.99   Vegetarian        No Mention   All Natural   0.06872
    Tomato Garden   2.29   Italian Sausage   Mushrooms    No Mention    0.06735
    Sundance        1.99   Vegetarian        No Mention   All Natural   0.06419

    Pregu           2.29   Meat              Mushrooms    All Natural   0.04137
    Pregu           2.49   Vegetarian        Mushrooms    No Mention    0.03578
    Sundance        2.29   Meat              No Mention   No Mention    0.03273
    Sundance        2.49   Italian Sausage   No Mention   No Mention    0.02081
    Tomato Garden   2.99   Italian Sausage   Mushrooms    No Mention    0.02055

    Sundance        2.79   Vegetarian        Mushrooms    No Mention    0.02022
    Tomato Garden   2.29   Vegetarian        No Mention   All Natural   0.01996
    Pregu           2.79   Italian Sausage   Mushrooms    No Mention    0.01233
    Pregu           2.49   Italian Sausage   No Mention   No Mention    0.01199
    Sundance        2.49   Meat              Mushrooms    All Natural   0.01010
    Sundance        2.99   Meat              Mushrooms    No Mention    0.00964
    Pregu           2.79   Meat              No Mention   No Mention    0.00763
    Pregu           2.99   Italian Sausage   No Mention   No Mention    0.00637
    Pregu           2.99   Meat              Mushrooms    All Natural   0.00547
    Tomato Garden   2.49   Vegetarian        Mushrooms    All Natural   0.00538

    Tomato Garden   2.79   Meat              Mushrooms    All Natural   0.00516
    Sundance        2.99   Vegetarian        No Mention   All Natural   0.00399
    Sundance        2.79   Meat              No Mention   All Natural   0.00266
    Tomato Garden   2.79   Vegetarian        No Mention   No Mention    0.00209
    Tomato Garden   2.99   Vegetarian        No Mention   No Mention    0.00162
    Tomato Garden   2.49   Meat              No Mention   No Mention    0.00088
```

The three methods produce different results.

Change in Market Share

The following steps simulate what would happen to the market if new products were introduced. Simulation observations are added to the data set and given zero weight. The conjoint analyses are rerun to compute the predicted utilities for the active observations and the simulations. The maximum utility model is used.

Recall that the design has numeric variables with values like 1, 2, and 3. Formats are used to display the descriptions of the levels of the attributes. The first thing we want to do is read in products to simulate. We could read in values like 1, 2, and 3 or we could read in more descriptive character values and convert them to numeric values using informats. We chose the latter approach. First we use PROC FORMAT to create the informats. Previously, we created formats with PROC FORMAT by specifying a `value` statement followed by pairs of the form *numeric-value=descriptive-character-string*. We create an informat with PROC FORMAT by specifying an `invalue` statement followed by pairs of the form *descriptive-character-string=numeric-value* as follows:

```
title 'Spaghetti Sauces';

proc format;
   invalue inbrand 'Preg'=1 'Sun' =2 'Tom' =3;
   invalue inmeat  'Veg' =1 'Meat'=2 'Ital'=3;
   invalue inmush  'Mush'=1 'No'  =2;
   invalue iningre 'Nat' =1 'No'  =2;
   invalue inprice '1.99'=1 '2.29'=2 '2.49'=3 '2.79'=4 '2.99'=5;
   run;
```

Next, we read the observations we want to consider for a sample market using the informats we just created. An `input` statement specification of the form "*variable* : *informat*" reads values starting with the first nonblank character. The following step creates the SAS data set:

```
data simulat;
   input brand       : inbrand.
         meat        : inmeat.
         mushroom    : inmush.
         ingredients : iningre.
         price       : inprice.;
   datalines;
Preg  Veg   Mush  Nat  1.99
Sun   Veg   Mush  Nat  1.99
Tom   Veg   Mush  Nat  1.99
Preg  Meat  Mush  Nat  2.49
Sun   Meat  Mush  Nat  2.49
Tom   Meat  Mush  Nat  2.49
Preg  Ital  Mush  Nat  2.79
Sun   Ital  Mush  Nat  2.79
Tom   Ital  Mush  Nat  2.79
;
```

Next, the original input data set is combined with the simulation observations. The subjects with poor fit are dropped and a `weight` variable is created to flag the simulation observations. The `weight` variable is not strictly necessary since all of the simulation observations have missing values on the ratings so they are excluded from the analysis that way. Still, it is good practice to explicitly use weights to exclude observations. The following steps process and display the data:

```
data inputdata2(drop=&droplist);
   set inputdata(in=w) simulat;
   Weight = w;
   run;

proc print;
   title2 'Simulation Observations Have a Weight of Zero';
   id weight;
   var brand -- price;
   run;
```

The results are as follows:

```
                              Spaghetti Sauces
                   Simulation Observations Have a Weight of Zero

  Weight    Brand            Meat                Mushroom     Ingredients    Price

     1      Pregu            Meat               No Mention    No Mention     2.79
     1      Tomato Garden    Vegetarian         No Mention    No Mention     2.79
     1      Pregu            Meat               Mushrooms     All Natural    2.29
     1      Tomato Garden    Vegetarian         Mushrooms     All Natural    2.49
     1      Sundance         Vegetarian         Mushrooms     No Mention     1.99
     1      Pregu            Italian Sausage    No Mention    No Mention     2.49
     1      Tomato Garden    Vegetarian         No Mention    No Mention     2.99
     1      Tomato Garden    Italian Sausage    Mushrooms     No Mention     2.29
     1      Pregu            Vegetarian         Mushrooms     No Mention     2.49
     1      Pregu            Vegetarian         No Mention    No Mention     2.29
     1      Sundance         Vegetarian         Mushrooms     No Mention     2.79
     1      Tomato Garden    Vegetarian         Mushrooms     No Mention     1.99
     1      Sundance         Meat               No Mention    No Mention     2.29
     1      Sundance         Meat               Mushrooms     No Mention     2.99
     1      Pregu            Italian Sausage    Mushrooms     No Mention     2.79
     1      Tomato Garden    Italian Sausage    Mushrooms     No Mention     2.99
     1      Sundance         Vegetarian         Mushrooms     All Natural    2.29
     1      Pregu            Meat               Mushrooms     All Natural    2.99
     1      Tomato Garden    Meat               No Mention    No Mention     2.49
     1      Sundance         Meat               Mushrooms     All Natural    2.49
     1      Pregu            Vegetarian         No Mention    All Natural    1.99
     1      Sundance         Meat               No Mention    All Natural    2.79
     1      Tomato Garden    Vegetarian         No Mention    All Natural    1.99
     1      Sundance         Italian Sausage    No Mention    No Mention     2.49
     1      Sundance         Vegetarian         No Mention    All Natural    1.99
     1      Sundance         Vegetarian         No Mention    All Natural    2.99
     1      Pregu            Italian Sausage    No Mention    No Mention     2.99
     1      Tomato Garden    Vegetarian         No Mention    All Natural    2.29
     1      Pregu            Vegetarian         Mushrooms     No Mention     1.99
     1      Tomato Garden    Meat               Mushrooms     All Natural    2.79
     0      Pregu            Vegetarian         Mushrooms     All Natural    1.99
     0      Sundance         Vegetarian         Mushrooms     All Natural    1.99
     0      Tomato Garden    Vegetarian         Mushrooms     All Natural    1.99
     0      Pregu            Meat               Mushrooms     All Natural    2.49
     0      Sundance         Meat               Mushrooms     All Natural    2.49
     0      Tomato Garden    Meat               Mushrooms     All Natural    2.49
     0      Pregu            Italian Sausage    Mushrooms     All Natural    2.79
     0      Sundance         Italian Sausage    Mushrooms     All Natural    2.79
     0      Tomato Garden    Italian Sausage    Mushrooms     All Natural    2.79
```

The next steps run the conjoint analyses suppressing the displayed output using the `noprint` option. The statement `weight weight` is specified since we want the simulation observations (which have zero weight) excluded from contributing to the analysis. However, the procedure still computes an expected utility for every observation including observations with zero, missing, and negative weights. The `outtest=` data set is created like before so we can check to make sure the *df* and R square look reasonable. The following steps perform the analysis and process and display the results:

```
ods exclude notes mvanova anova;
proc transreg data=inputdata2 utilities short noprint
   separators=', ' lprefix=0 method=morals outtest=utils;
   title2 'Conjoint Analysis';
   model identity(sub:) =
         class(brand | price meat mushroom ingredients / zero=sum);
   output p ireplace out=results3 coefficients;
   weight weight;
   run;

data model;
   set utils;
   if statistic in ('R-Square', 'Adj R-Sq', 'Model');
   Subj = scan(_depvar_, 2);
   if statistic = 'Model' then do;
      value = numdf;
      statistic = 'Num DF';
      output;
      value = dendf;
      statistic = 'Den DF';
      output;
      value = dendf + numdf + 1;
      statistic = 'N';
      end;
   output;
   keep statistic value subj;
   run;

proc transpose data=model out=summ;
   by subj;
   idlabel statistic;
   id statistic;
   run;

proc print label data=summ(drop=_name_ _label_); run;
```

The following SAS log messages tell us that the nine simulation observations were deleted both because of zero weight and because of missing values in the dependent variables.

```
NOTE: 9 observations were deleted from the analysis but not from the
      output data set due to missing values.
NOTE: 9 observations were deleted from the analysis but not from the
      output data set due to nonpositive weights.
NOTE: A total of 9 observations were deleted.
```

The *df* and R square results, some of which are shown next, look fine:

```
                           Spaghetti Sauces
                          Conjoint Analysis

                          Num     Den                            Adj
           Obs     Subj     DF      DF      N     R-Square       R-Sq

             1    Sub001    18      11     30      0.83441    0.56345
             2    Sub002    18      11     30      0.91844    0.78497
             3    Sub003    18      11     30      0.92908    0.81302
             .
             .
             .
            81    Sub099    18      11     30      0.88920    0.70789
            82    Sub100    18      11     30      0.90330    0.74507
```

In the following steps, the simulation observations are pulled out of the `out=` data set, and the %SIM macro is run to simulate market share:

```
data results4;
   set results3;
   where weight = 0;
   run;

%sim(data=results4, out=shares2, method=max,
     idvars=price brand meat mushroom ingredients);
```

The results are as follows:

```
                           Spaghetti Sauces
                         Expected Market Share
                         Maximum Utility Model

Brand            Price    Meat                 Mushroom     Ingredients     Share

Pregu            1.99     Vegetarian           Mushrooms    All Natural    0.35976
Sundance         1.99     Vegetarian           Mushrooms    All Natural    0.29878
Tomato Garden    1.99     Vegetarian           Mushrooms    All Natural    0.19512
Tomato Garden    2.79     Italian Sausage      Mushrooms    All Natural    0.08537
Sundance         2.79     Italian Sausage      Mushrooms    All Natural    0.02439
Pregu            2.49     Meat                 Mushrooms    All Natural    0.01220
Sundance         2.49     Meat                 Mushrooms    All Natural    0.01220
Pregu            2.79     Italian Sausage      Mushrooms    All Natural    0.01220
Tomato Garden    2.49     Meat                 Mushrooms    All Natural    0.00000
```

For this set of products, the inexpensive vegetarian sauces have the greatest market share with Pregu brand preferred over Sundance and Tomato Garden. Now we'll consider adding six more products to the market, the six meat sauces we just saw, but at a lower price. The following steps create the data set, and perform the analysis:

```
data simulat2;
   input brand       : inbrand.
         meat        : inmeat.
         mushroom    : inmush.
         ingredients : iningre.
         price       : inprice.;
   datalines;
Preg Meat Mush Nat 2.29
Sun  Meat Mush Nat 2.29
Tom  Meat Mush Nat 2.29
Preg Ital Mush Nat 2.49
Sun  Ital Mush Nat 2.49
Tom  Ital Mush Nat 2.49
;

data inputdata3(drop=&droplist);
   set inputdata(in=w) simulat simulat2;
   weight = w;
   run;

ods exclude notes mvanova anova;
proc transreg data=inputdata3 utilities short noprint
   separators=', ' lprefix=0 method=morals outtest=utils;
   title2 'Conjoint Analysis';
   model identity(sub:) =
         class(brand | price meat mushroom ingredients / zero=sum);
   output p ireplace out=results5 coefficients;
   weight weight;
   run;
```

The following notes tell us that 15 simulation observations were excluded:

```
NOTE: 15 observations were deleted from the analysis but not from the
      output data set due to missing values.
NOTE: 15 observations were deleted from the analysis but not from the
      output data set due to nonpositive weights.
NOTE: A total of 15 observations were deleted.
```

The following steps extract the *df* and R square and display the results:

```
data model;
   set utils;
   if statistic in ('R-Square', 'Adj R-Sq', 'Model');
   Subj = scan(_depvar_, 2);
   if statistic = 'Model' then do;
      value = numdf;
      statistic = 'Num DF';
      output;
      value = dendf;
      statistic = 'Den DF';
      output;
      value = dendf + numdf + 1;
      statistic = 'N';
      end;
   output;
   keep statistic value subj;
   run;

proc transpose data=model out=summ;
   by subj;
   idlabel statistic;
   id statistic;
   run;

proc print label data=summ(drop=_name_ _label_); run;
```

The results are as follows:

```
                          Spaghetti Sauces
                         Conjoint Analysis

                          Num    Den                        Adj
        Obs      Subj      DF     DF      N    R-Square     R-Sq

          1     Sub001     18     11     30     0.83441    0.56345
          2     Sub002     18     11     30     0.91844    0.78497
          3     Sub003     18     11     30     0.92908    0.81302
          .
          .
          .
         81     Sub099     18     11     30     0.88920    0.70789
         82     Sub100     18     11     30     0.90330    0.74507
```

The *df* and R square still look fine.

The following steps run the simulation with all 15 simulation observations:

```
data results6;
   set results5;
   where weight = 0;
   run;

%sim(data=results6, out=shares3, method=max,
     idvars=price brand meat mushroom ingredients);
```

The results are as follows:

```
                              Spaghetti Sauces
                            Expected Market Share
                            Maximum Utility Model

Brand            Price    Meat                Mushroom     Ingredients     Share

Sundance         1.99     Vegetarian          Mushrooms    All Natural    0.25813
Pregu            1.99     Vegetarian          Mushrooms    All Natural    0.20935
Pregu            2.29     Meat                Mushrooms    All Natural    0.19512
Tomato Garden    1.99     Vegetarian          Mushrooms    All Natural    0.15447
Sundance         2.49     Italian Sausage     Mushrooms    All Natural    0.08537
Sundance         2.29     Meat                Mushrooms    All Natural    0.03659
Tomato Garden    2.49     Italian Sausage     Mushrooms    All Natural    0.01829
Tomato Garden    2.29     Meat                Mushrooms    All Natural    0.01220
Pregu            2.49     Italian Sausage     Mushrooms    All Natural    0.01220
Tomato Garden    2.79     Italian Sausage     Mushrooms    All Natural    0.01220
Sundance         2.79     Italian Sausage     Mushrooms    All Natural    0.00610
Pregu            2.49     Meat                Mushrooms    All Natural    0.00000
Sundance         2.49     Meat                Mushrooms    All Natural    0.00000
Tomato Garden    2.49     Meat                Mushrooms    All Natural    0.00000
Pregu            2.79     Italian Sausage     Mushrooms    All Natural    0.00000
```

The following steps merge the data set containing the old market shares with the data set containing the new market shares to show the effect of adding the new products:

```
title 'Spaghetti Sauces';

proc sort data=shares2;
   by price brand meat mushroom ingredients;
   run;

proc sort data=shares3;
   by price brand meat mushroom ingredients;
   run;
```

```
data both;
   merge shares2(rename=(share=OldShare)) shares3;
   by price brand meat mushroom ingredients;
   if oldshare = . then Change = 0;
   else change = oldshare;
   change = share - change;
   run;

proc sort;
   by descending share price brand meat mushroom ingredients;
   run;

options missing=' ';
proc print noobs;
   title2 'Expected Market Share and Change';
   var price brand meat mushroom ingredients
       oldshare share change;
   format oldshare -- change 6.3;
   run;
options missing=.;
```

The results are as follows:

```
                              Spaghetti Sauces
                       Expected Market Share and Change

                                                               Old
 Price Brand         Meat            Mushroom  Ingredients   Share   Share Change

  1.99 Sundance      Vegetarian      Mushrooms All Natural   0.299   0.258 -0.041
  1.99 Pregu         Vegetarian      Mushrooms All Natural   0.360   0.209 -0.150
  2.29 Pregu         Meat            Mushrooms All Natural           0.195  0.195
  1.99 Tomato Garden Vegetarian      Mushrooms All Natural   0.195   0.154 -0.041
  2.49 Sundance      Italian Sausage Mushrooms All Natural           0.085  0.085
  2.29 Sundance      Meat            Mushrooms All Natural           0.037  0.037
  2.49 Tomato Garden Italian Sausage Mushrooms All Natural           0.018  0.018
  2.29 Tomato Garden Meat            Mushrooms All Natural           0.012  0.012
  2.49 Pregu         Italian Sausage Mushrooms All Natural           0.012  0.012
  2.79 Tomato Garden Italian Sausage Mushrooms All Natural   0.085   0.012 -0.073
  2.79 Sundance      Italian Sausage Mushrooms All Natural   0.024   0.006 -0.018
  2.49 Pregu         Meat            Mushrooms All Natural   0.012   0.000 -0.012
  2.49 Sundance      Meat            Mushrooms All Natural   0.012   0.000 -0.012
  2.49 Tomato Garden Meat            Mushrooms All Natural   0.000   0.000  0.000
  2.79 Pregu         Italian Sausage Mushrooms All Natural   0.012   0.000 -0.012
```

We see that the vegetarian sauces are most preferred, but we predict they would lose share if the new meat sauces were entered in the market. In particular, the Sundance and Pregu meat sauces would gain significant market share under this model.

PROC TRANSREG Specifications

PROC TRANSREG (transformation regression) is used to perform conjoint analysis and many other types of analyses, including simple regression, multiple regression, redundancy analysis, canonical correlation, analysis of variance, and external unfolding, all with nonlinear transformations of the variables. This section documents the statements and options available in PROC TRANSREG that are commonly used in conjoint analyses. See "The TRANSREG Procedure" in the *SAS/STAT User's Guide* for more information about PROC TRANSREG. This section documents only a small subset of the capabilities of PROC TRANSREG.

The following statements are used in the TRANSREG procedure for conjoint analysis:

```
PROC TRANSREG <DATA=SAS-data-set> <OUTTEST=SAS-data-set>
              <a-options> <o-options>;
   MODEL transform(dependents   </ t-options>) =
         transform(independents </ t-options>)
        <transform(independents </ t-options>) ...> </ a-options>;
   OUTPUT <OUT=SAS-data-set> <o-options>;
   WEIGHT variable;
   ID variables;
   BY variables;
```

Specify the `proc` and `model` statements to use PROC TRANSREG. The `output` statement is required to produce an `out=` output data set, which contains the transformations, indicator variables, and predicted utility for each product. The `outtest=` data set, which contains the ANOVA, regression, and part-worth utility tables, is requested in the `proc` statement. All options can be abbreviated to their first three letters.

PROC TRANSREG *Statement*

```
PROC TRANSREG <DATA=SAS-data-set> <OUTTEST=SAS-data-set>
              <a-options> <o-options>;
```

The `data=` and `outtest=` options can appear only in the PROC TRANSREG statement. The algorithm options *(a-options)* appear in the `proc` or `model` statement. The output options *(o-options)* can appear in the `proc` or `output` statement.

`DATA=`*SAS-data-set*
specifies the input SAS data. If the `data=` option is not specified, PROC TRANSREG uses the most recently created SAS data set.

`OUTTEST=`*SAS-data-set*
specifies an output data set that contains the ANOVA table, R square, and the conjoint analysis part-worth utilities, and the attribute importances.

Algorithm Options

```
PROC TRANSREG <DATA=SAS-data-set> <OUTTEST=SAS-data-set>
              <a-options> <o-options>;
   MODEL transform(dependents   </ t-options>) =
         transform(independents </ t-options>)
        <transform(independents </ t-options>) ...> </ a-options>;
```

Algorithm options can appear in the `proc` or `model` statement as *a-options*.

CONVERGE=n
specifies the minimum average absolute change in standardized variable scores that is required to continue iterating. By default, `converge=0.00001`.

DUMMY
requests a canonical initialization. When `spline` transformations are requested, specify `dummy` to solve for the optimal transformations without iteration. Iteration is only necessary when there are monotonicity constraints.

LPREFIX=n
specifies the number of first characters of a `class` variable's label (or name if no label is specified) to use in constructing labels for part-worth utilities. For example, the default label for `Brand=Duff` is "Brand Duff". If you specify `lprefix=0` then the label is simply "Duff".

MAXITER=n
specifies the maximum number of iterations. By default, `maxiter=30`.

NOPRINT
suppresses the display of all output.

ORDER=FORMATTED
ORDER=INTERNAL
specifies the order in which the CLASS variable levels are reported. The default, `order=internal`, sorts by unformatted value. Specify `order=formatted` when you want the levels sorted by formatted value. Sort order is machine dependent. Note that in Version 6 and Version 7 of the SAS System, the default sort order was `order=formatted`. The default was changed to `order=internal` in Version 8 to be consistent with Base SAS procedures.

METHOD=MORALS
METHOD=UNIVARIATE
specifies the iterative algorithm. Both `method=morals` and `method=univariate` fit univariate multiple regression models with the possibility of nonlinear transformations of the variables. They differ in the way they structure the output data set when there is more than one dependent variable. When it can be used, `method=univariate` is more efficient than `method=morals`.

You can use `method=univariate` when no transformations of the independent variables are requested, for example, when the independent variables are all designated `class`, `identity`, or `pspline`. In this case, the final set of independent variables is the same for all subjects. If transformations such as `monotone`, `identity`, `spline` or `mspline` are specified for the independent variables, the transformed independent variables may be different for each dependent variable and so must be output separately for each dependent variable. In conjoint analysis, there is typically one dependent variable for each subject. This is illustrated in the examples.

With `method=univariate` and more than one dependent variable, PROC TRANSREG creates a data set with the same number of score observations as the original but with more variables. The untransformed dependent variable names are unchanged. The default transformed dependent variable names consist of the prefix "T" and the original variable names. The default predicted value names consist of the prefix "P" and the original variable names. The full set of independent variables appears once.

When more than one dependent variable is specified, `method=morals` creates a *rolled-out* data set with the dependent variable in `_depend_`, its transformation in `t_depend_`, and its predicted values in `p_depend_`. The full set of independents is repeated for each (original) dependent variable.

The procedure chooses a default method based on what is specified in the `model` statement. When transformations of the independent variables are requested, the default method is `morals`. Otherwise the default method is `univariate`.

`SEPARATORS=`*string-1 <string-2 >*
specifies separators for creating labels for the part-worth utilities. By default, `separators=' ' ' * '` ("blank" and "blank asterisk blank"). The first value is used to separate variable names and values in interactions. The second value is used to separate interaction components. For example, the default label for `Brand=Duff` is "Brand Duff". If you specify `separators=', '` then the label is "Brand, Duff". Furthermore, the default label for the interaction of `Brand=Duff` and `Price=3.99` is "Brand Duff * Price 3.99". You could specify `lprefix=0` and `separators='' ' @ '` to instead create labels like "Duff @ 3.99". You use the `lprefix=0` option when you want to construct labels using zero characters of the variable name, that is when you want to construct labels from just the formatted level. The option `separators='' ' @ '` specifies in the second string a separator of the form "blank at blank". In this case, the first string is ignored because with `lprefix=0` there is no name to separate from the level.

`SHORT`
suppresses the iteration histories. For most standard metric conjoint analyses, no iterations are necessary, so specifying `short` eliminates unnecessary output. PROC TRANSREG displays a message if it ever fails to converge, so it is usually safe to specify the `short` option.

`UTILITIES`
displays the part-worth utilities and importances table and an ANOVA table. Note that you can use an `ods exclude` statement to exclude ANOVA tables and unnecessary notes from the conjoint output (see page 684).

Output Options

```
    PROC TRANSREG <DATA=SAS-data-set> <OUTTEST=SAS-data-set>
                  <a-options> <o-options>;
       OUTPUT <OUT=SAS-data-set> <o-options>;
```

The `out=` option can only appear in the `output` statement. The other output options can appear in the `proc` or `output` statement as *o-options*.

`COEFFICIENTS`
outputs the part-worth utilities to the `out=` data set.

`P`
includes the predicted values in the `out=` output data set, which are the predicted utilities for each product. By default, the predicted values variable name is the original dependent variable name prefixed with a "P".

`IREPLACE`
replaces the original independent variables with the transformed independent variables in the output data set. The names of the transformed variables in the output data set correspond to the names of the original independent variables in the input data set.

`OUT=`*SAS-data-set*
names the output data set. When an `output` statement is specified without the `out=` option, PROC TRANSREG creates a data set and uses the DATA*n* convention. To create a permanent SAS data set, specify a two-level name. The data set contains the original input variables, the coded indicator variables, the transformation of the dependent variable, and the optionally predicted utilities for each product.

`RESIDUALS`
outputs to the `out=` data set the differences between the observed and predicted utilities. By default, the residual variable name is the original dependent variable name prefixed with an "R".

Transformations and Expansions

```
MODEL transform(dependents   </ t-options>) =
      transform(independents </ t-options>)
     <transform(independents </ t-options>) ...> </ a-options>;
```

The operators "`*`", "`|`", and "`@`" from the GLM procedure are available for interactions with `class` variables.

```
class(a * b ...
      c | d ...
      e | f ... @ n)
```

For example, the following statement fits 100 individual main-effects models:

```
model identity(rating1-rating100) = class(x1-x5 / zero=sum);
```

The following statement fits models with main effects and all two-way interactions:

```
model identity(rating1-rating100) = class(x1|x2|x3|x4|x5@2 / zero=sum);
```

The following statement fits models with main effects and some two-way interactions:

```
   model identity(rating1-rating100) = class(x1-x5 x1*x2 x3*x4 / zero=sum);
```

You can also fit separate price functions within each brand by specifying the following:

```
   model identity(rating1-rating100) =
         class(brand / zero=none) | spline(price);
```

The list `x1-x5` is equivalent to `x1 x2 x3 x4 x5`. The vertical bar specifies all main effects and interactions, and the at sign limits the interactions. For example, `@2` limits the model to main effects and two-way interactions. The list `x1|x2|x3|x4|x5@2` is equivalent to `x1 x2 x1 * x2 x3 x1 * x3 x2 * x3 x4 x1 * x4 x2 * x4 x3 * x4 x5 x1 * x5 x2 * x5 x3 * x5 x4 * x5`. The specification `x1 * x2` indicates the two-way interaction between `x1` and `x2`, and `x1 * x2 * x3` indicates the three-way interaction between `x1`, `x2`, and `x3`.

Each of the following can be specified in the `model` statement as a *transform*. The `pspline` and `class` expansions create more than one output variable for each input variable. The rest are transformations that create one output variable for each input variable.

`CLASS`
designates variables for analysis as nominal-scale-of-measurement variables. For conjoint analysis, the `zero=sum` *t-option* is typically specified: `class(`*variables* `/ zero=sum)`. Variables designated as `class` variables are expanded to a set of indicator variables. Usually the number output variables for each `class` variable is the number of different values in the input variables. Dependent variables should not be designated as `class` variables.

`IDENTITY`
variables are not changed by the iterations. The `identity(`*variables*`)` specification designates interval-scale-of-measurement variables when no transformation is permitted. When small data values mean high preference, you need to use the `reflect` transformation option.

`MONOTONE`
monotonically transforms variables; ties are preserved. When `monotone(`*variables*`)` is used with dependent variables, a nonmetric conjoint analysis is performed. When small data values mean high preference, you need to use the `reflect` transformation option. The `monotone` specification can also be used with independent variables to impose monotonicity on the part-worth utilities. When it is known that monotonicity should exist in an attribute variable, using `monotone` instead of `class` for that attribute may improve prediction. An option exists in PROC TRANSREG for optimally untying tied values, but this option should not be used because it almost always produces a degenerate result.

`MSPLINE`
monotonically and smoothly transforms variables. By default, `mspline(`*variables*`)` fits a monotonic quadratic spline with no knots. Knots are specified as *t-options*, for example, `mspline(`*variables* `/ nknots=3)` or `mspline(`*variables* `/ knots=5 to 15 by 5)`. Like `monotone`, `mspline`, finds a monotonic transformation. Unlike `monotone`, `mspline` places a bound on the *df* (number of knots + degree) used by the transformation. With `mspline`, it is possible to allow for nonlinearity in the responses and still have error *df*. This is not always possible with `monotone`. When small data values mean high preference, you need to use the `reflect` transformation option. You can also use `mspline` with attribute variables to impose monotonicity on the part-worth utilities.

`PSPLINE`
expands each variable to a piece-wise polynomial spline basis. By default, `pspline(`*variables*`)` uses a cubic spline with no knots. Knots are specified as *t-options.* Specify `pspline(`*variable* `/ degree=2)` for an attribute variable to fit a quadratic model. For each `pspline` variable, $d + k$ output variables are created, where d is the degree of the polynomial and k is the number of knots. You should not specify `pspline` with the dependent variables.

`RANK`
performs a rank transformation, with ranks averaged within ties. Rating-scale data can be transformed to ranks by specifying `rank(`*variables*`)`. When small data values mean high preference, you need to use the `reflect` transformation option. Typically, `rank` is only used for dependent variables. For example, if a rating-scale variable has sorted values 1, 1, 1, 2, 3, 3, 4, 5, 5, 5, then the rank transformation is 2, 2, 2, 4, 5.5, 5.5, 7, 9, 9, 9. A conjoint analysis of the original rating-scale variable is usually not the same as a conjoint analysis of a rank transformation of the ratings. With ordinal-scale-of-measurement data, it is often good to analyze rank transformations instead of the original data. An alternative is to specify `monotone`, which performs a nonmetric conjoint analysis. For real data, `monotone` always finds a better fit than `rank`, but `rank` may lead to better prediction.

`SPLINE`
smoothly transforms variables. By default, `spline(`*variables*`)` fits a cubic spline with no knots. Knots are specified as *t-options.* Like `pspline`, `spline` models nonlinearities in the attributes.

Transformation Options

```
MODEL transform(dependents   </ t-options>) =
      transform(independents </ t-options>)
     <transform(independents </ t-options>) ...> </ a-options>;
```

The following are specified in the `model` statement as *t-options*'s.

`DEGREE=`n
specifies the degree of the spline. The defaults are `degree=3` (cubic spline) for `spline` and `pspline`, and `degree=2` (quadratic spline) for `mspline`. For example, to request a quadratic spline, specify `spline(`*variables* `/ degree=2)`.

`EVENLY`
is used with the `nknots=` option to evenly space the knots for splines. For example, if `spline(x / nknots=2 evenly)` is specified and `x` has a minimum of 4 and a maximum of 10, then the two interior knots are 6 and 8. Without `evenly`, the `nknots=` option places knots at percentiles, so the knots are not evenly spaced.

`KNOTS=`*numberlist*
specifies the interior knots or break points for splines. By default, there are no knots. For example, to request knots at 1, 2, 3, 4, 5, specify `spline(`*variable* `/ knots=1 to 5)`.

`NKNOTS=`*k*
creates *k* knots for splines: the first at the $100/(k+1)$ percentile, the second at the $200/(k+1)$ percentile, and so on. Unless `evenly` is specified, knots are placed at data values; there is no interpolation. For example, with `spline(`*variable* `/ NKNOTS=3)`, knots are placed at the twenty-fifth percentile, the median, and the seventy-fifth percentile. By default, `nknots=0`.

`REFLECT`
reflects the transformation around its mean, $Y = -(Y - Y) + Y$, after the iterations are completed and before the final standardization and results calculations. This option is particularly useful with the dependent variable. When the dependent variable consists of ranks with the most preferred combination assigned 1.0, `identity(`*variable* `/ reflect)` reflects the transformation so that positive utilities mean high preference.

`ZERO=SUM`
constrains the part-worth utilities to sum to zero within each attribute. The specification `class(`*variables* `/ zero=sum)` creates a less than full rank model, but the coefficients are uniquely determined due to the sum-to-zero constraint.

BY *Statement*

```
    BY variables;
```

A `by` statement can be used with PROC TRANSREG to obtain separate analyses on observations in groups defined by the `by` variables. When a `by` statement appears, the procedure expects the input data set to be sorted in order of the `by` variables.

If the input data set is not sorted in ascending order, use one of the following alternatives:

- Use the SORT procedure with a similar `by` statement to sort the data.
- Use the `by` statement options `notsorted` or `descending` in the `by` statement for the TRANSREG procedure. As a cautionary note, the `notsorted` option does not mean that the data are unsorted. It means that the data are arranged in groups (according to values of the `by` variables), and these groups are not necessarily in alphabetical or increasing numeric order.
- Use the DATASETS procedure (in base SAS software) to create an index on the `by` variables.

For more information about the by statement, see the discussion in *SAS Language: Reference*. For more information about the DATASETS procedure, see the discussion in *SAS Procedures Guide*.

ID *Statement*

```
ID variables;
```

The id statement includes additional character or numeric variables from the input data set in the out= data set.

WEIGHT *Statement*

```
WEIGHT variable;
```

A weight statement can be used in conjoint analysis to distinguish ordinary active observations, holdouts, and simulation observations. When a weight statement is used, a weighted residual sum of squares is minimized. The observation is used in the analysis only if the value of the weight statement variable is greater than zero. For observations with positive weight, the weight statement has no effect on *df* or number of observations, but the weights affect most other calculations.

Assign each active observation a weight of 1. Assign each holdout observation a weight that excludes it from the analysis, such as missing. Assign each simulation observation a different weight that excludes it from the analysis, such as zero. Holdouts are rated by the subjects and so have nonmissing values in the dependent variables. Simulation observations are not rated and so have missing values in the dependent variable. It is useful to create a format for the weight variable that distinguishes the three types of observations in the input and output data sets, for example, as follows:

```
proc format;
   value wf 1  = 'Active'
            .  = 'Holdout'
            0  = 'Simulation';
   run;
```

PROC TRANSREG does not distinguish between weights that are zero, missing, or negative. All nonpositive weights exclude the observations from the analysis. The holdout and simulation observations are given different nonpositive values and a format to make them easy to distinguish in subsequent analyses and listings. The part-worth utilities for each attribute are computed using only those observations with positive weight. The predicted utility is computed for all products, even those with nonpositive weights.

Monotone, Spline, and Monotone Spline Comparisons

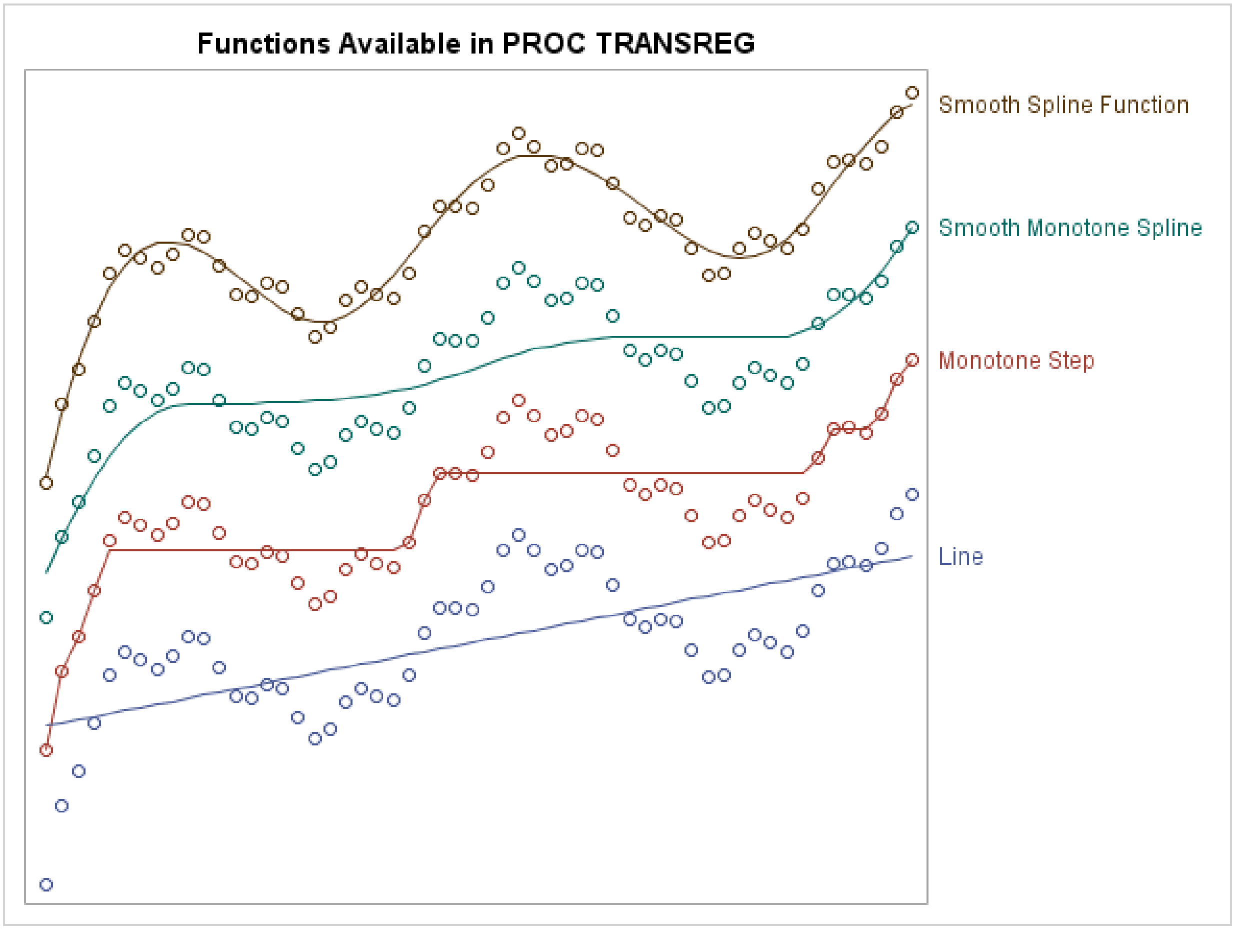
When you choose the transformation of the ratings or rankings, you choose among

identity - model the data directly

monotone - model an increasing step function of the data `mspline` - model a nonlinear but smooth and increasing function of the data

`spline` - model a smooth function of the data

The following plot shows examples of the different types of functions you can fit in PROC TRANSREG. In each case, a function is fit to the same artificial nonlinear data. The top function is a spline function, created by `spline`. It is smooth and nonlinear. It follows the overall shape of the data, but smooths out the smaller bumps. Below that is a monotone spline function, created by `mspline`. Like the spline function, it is smooth and nonlinear. Unlike the spline function, it is monotonic. The function never decreases; it always rises or stays flat. The monotone spline function follows the overall upward trend in the data, and it shows the changes in upward trend, but it smooths out all the dips and bumps in the function. Below the monotone spline function is a monotone step function, created by `monotone`. It is not smooth, but it is monotonic. Like the monotone spline, the monotone step function follows the overall upward trend in the data, and it smooths out all the dips and bumps in the function. However, the function is not smooth, and it typically requires many more parameters be fit than with monotone splines. Below the monotone step function is a line, created by `identity`. It is smooth and linear. It follows the overall upward trend in the data, but it smooths over all the dips, bumps, and changes in upward trend.



Typical conjoint analyses are metric (using `identity`) or nonmetric (using `monotone`). While not often used in practice, monotone splines have a lot to recommend them. They allow for nonlinearities in the transformation of preference, but unlike `monotone`, they are smooth and do not use up all of your error *df*. One would typically never use `spline` on the ratings or rankings in a conjoint analysis, but if for some reason, you had a lot of price points,[*] you could fit a spline function of the price attribute. This would allow for nonlinearities in preferences for different prices while constraining the part-worth utility function to be smooth.

[*]For design efficiency reasons, you typically should not.

Samples of PROC TRANSREG Usage

Conjoint analysis can be performed in many ways with PROC TRANSREG. This section provides sample specifications for some typical and some more esoteric conjoint analyses. The dependent variables typically contain ratings or rankings of products by a number of subjects. The independent variables, `x1-x5`, are the attributes. For metric conjoint analysis, the dependent variable is designated `identity`. For nonmetric conjoint analysis, `monotone` is used. Attributes are usually designated as `class` variables with the restriction that the part-worth utilities within each attribute sum to zero.

The `utilities` option requests an overall ANOVA table, a table of part-worth utilities, their standard errors, and the importance of each attribute. The `p` (predicted values) option outputs to a data set the predicted utility for each product. The `ireplace` option suppresses the separate output of transformed independent variables since the independent variable transformations are the same as the raw independent variables. The `weight` variable is used to distinguish active observations from holdouts and simulation observations. The `reflect` transformation option reflects the transformation of the ranking so that large transformed values, positive utility, and positive evaluation all correspond.

Today, metric conjoint analysis is used more often than nonmetric conjoint analysis, and rating-scale data are collected more often than rankings.

Metric Conjoint Analysis with Rating-Scale Data

The following step performs a metric conjoint analysis with rating-scale data:

```
ods exclude notes mvanova anova;
proc transreg data=a utilities short method=morals;
   model identity(rating1-rating100) = class(x1-x5 / zero=sum);
   output p ireplace;
   weight w;
   run;
```

Nonmetric Conjoint Analysis

The following step performs a nonmetric conjoint analysis specification, which has *many* parameters for the transformations:

```
ods exclude notes anova liberalanova conservanova
            mvanova liberalmvanova conservmvanova;
proc transreg data=a utilities short maxiter=500 method=morals;
   model monotone(ranking1-ranking100 / reflect) = class(x1-x5 / zero=sum);
   output p ireplace;
   weight w;
   run;
```

Monotone Splines

The following step performs a conjoint analysis that is more restrictive than a nonmetric analysis but less restrictive than a metric conjoint analysis:

```
ods exclude notes anova liberalanova conservanova
            mvanova liberalmvanova conservmvanova;
proc transreg data=a utilities short maxiter=500 method=morals;
   model mspline(ranking1-ranking100 / reflect) =
         class(x1-x5 / zero=sum);
   output p ireplace;
   weight w;
   run;
```

By default, the monotone spline transformation has two parameters (degree two with no knots). If less smoothness is desired, specify knots, for example, as follows:

```
ods exclude notes anova liberalanova conservanova
            mvanova liberalmvanova conservmvanova;
proc transreg data=a utilities short maxiter=500 method=morals;
   model mspline(ranking1-ranking100 / reflect nknots=3) =
         class(x1-x5 / zero=sum);
   output p ireplace;
   weight w;
   run;
```

Each knot requires estimation of an additional parameter.

Constraints on the Utilities

The following step performs a metric conjoint analysis with linearity constraints imposed on `x4` and monotonicity constraints imposed on `x5`.

```
ods exclude notes anova liberalanova conservanova
            mvanova liberalmvanova conservmvanova
            liberalutilities liberalfitstatistics;
proc transreg data=a utilities short maxiter=500 method=morals;
   model identity(rating1-rating100) = class(x1-x3 / zero=sum)
         identity(x4) monotone(x5);
   output p ireplace;
   weight w;
   run;
```

With the monotonic constraints on the part-worth utilities, PROC TRANSREG displays some extra information, liberal and conservative part-worth utility and fit statistics tables. These tables report the same part-worth utilities, but they are based on different methods of counting the number of parameters estimated. The liberal test tables can be suppressed by adding `liberalutilities liberalfitstatistics` to the `ods exclude` statement.

The following step performs specifies a monotonic step-function constraint on `x1-x5` and a smooth, monotonic transformation of `price`:

```
ods exclude notes anova liberalanova conservanova
            mvanova liberalmvanova conservmvanova
            liberalutilities liberalfitstatistics;
proc transreg data=a utilities short maxiter=500 method=morals;
   model identity(rating1-rating100) = monotone(x1-x5) mspline(price);
   output p ireplace;
   weight w;
   run;
```

A Discontinuous Price Function

The utility of price may not be a continuous function of price. It has been frequently found that utility is discontinuous at *round* numbers such as $1.00, $2.00, $100, $1000, and so on. If `price` has many values in the data set, say over the range $1.05 to $3.95, then a monotone function of price with discontinuities at $2.00 and $3.00 can be requested as follows:

```
ods exclude notes anova liberalanova conservanova
            mvanova liberalmvanova conservmvanova
            liberalutilities liberalfitstatistics;
proc transreg data=a utilities short maxiter=500 method=morals;
   model identity(rating1-rating100) =
         class(x1-x5 / zero=sum)
         mspline(price / knots=2 2 2 3 3 3);
   output p ireplace;
   weight w;
   run;
```

The monotone spline is degree two. The *order* of the spline is one greater than the degree; in this case the order is three. When the same knot value is specified *order* times, the transformation is discontinuous at the knot. See page 1213, for some applications of splines to conjoint analysis.

The Macros

Warren F. Kuhfeld

Abstract

SAS provides a set of macros for designing experiments and analyzing choice data. The syntax and usage of these macros is discussed in this chapter. Two additional macros, one for scatter plots of labeled points and one for color interpolation, are documented here as well.*

Introduction

The following SAS autocall macros are available:

Macro	Page	Required Products	Purpose
`%ChoicEff`	806	STAT, IML	efficient choice design
`%MktAllo`	956		processes allocation data
`%MktBal`	959	QC	balanced main-effects designs
`%MktBIBD`	963	IML, QC	balanced incomplete block designs
`%MktBlock`	979	STAT, IML	block a linear or choice design
`%MktBSize`	989		sizes of balanced incomplete block designs
`%MktDes`	995	STAT, QC	efficient factorial design via candidate set search
`%MktDups`	1004		identify duplicate choice sets or runs
`%MktEval`	1012	STAT, IML	evaluate an experimental design
`%MktEx`	1017	STAT, IML, QC	efficient factorial design
`%MktKey`	1090		aid creation of the `key=` data set
`%MktLab`	1093		relabel, rename and assign levels to design factors
`%MktMDiff`	1105	STAT	MaxDiff (best-worst) analysis
`%MktMerge`	1125		merges a choice design with choice data
`%MktOrth`	1128		lists the orthogonal array catalog
`%MktPPro`	1145	IML	partial profiles through BIB designs
`%MktRoll`	1153		rolls a factorial design into a choice design
`%MktRuns`	1159		selecting number of runs in an experimental design
`%Paint`	1169		color interpolation
`%PHChoice`	1173		customizes the output from a choice model
`%PlotIt`	1178	STAT, GRAPH	graphical scatter plots of labeled points

*Copies of this chapter (MR-2010I), the other chapters, sample code, and all of the macros are available on the Web `http://support.sas.com/resources/papers/tnote/tnote_marketresearch.html`. Specifically, sample code is here `http://support.sas.com/techsup/technote/mr2010i.sas`. For help, please contact SAS Technical Support. See page 25 for more information.

These macros[*] are used for generating efficient factorial designs, efficient choice designs, processing and evaluating designs, processing data from choice experiments, and for certain graphical displays. The BASE product is required to run all macros along with any additional indicated products. To run the full suite of macros you need to have BASE, SAS/STAT, SAS/QC and SAS/IML installed, and you need SAS/GRAPH for the plotting macro. Once you have the right products installed, and you have installed the macros, you can call them and use them just as you would use a SAS procedure. However, the syntax for macros is a little different from SAS procedures. The following step makes an experimental design with 1 two-level and 7 three-level factors with 18 runs or profiles:

```
   %mktex(2 3 ** 7, n=18)
```

Changes and Enhancements

PROC TRANSREG has a new `sta` or `standorth` option for standardized orthogonal contrast coding that can be used with the `%ChoicEff` macro. This along with the new option, `options=relative`, can be used to get a relative *D*-efficiency in the 0 to 100 range for certain choice designs. The TRANSREG option is first available with SAS 9.2. It will not be recognized, and it will cause an error in earlier SAS releases.

The new `%MktBIBD` macro for balanced incomplete block designs (BIBDs) was added, and a number of the examples were revised to use it. This macro can also make BIBDs with directional and nondirectional row-neighbor balance. The new `%MktMDiff` macro analyzes data from MaxDiff (best-worst) choice models. Some new orthogonal arrays were added to the `%MktEx` macro, and there are a few other changes as well. There is a new discussion of Latin square and Graeco-Latin Square designs beginning on page 1026.

Installation

If your site has installed the autocall libraries supplied by SAS and uses the standard configuration of SAS supplied software, you need to ensure that the SAS system option `mautosource` is in effect to begin using the autocall macros. Note, however, that there might be differences between the macros used in this documentation and those that were shipped with your version of SAS. Be sure to get the latest macros from `http://support.sas.com/resources/papers/tnote/tnote_marketresearch.html`. These macros will work with Version 9 and later versions of SAS. You should install *all* of these macros, not just one or some. Some of the macros call other macros and will not work if the other macros are not there or if only older versions of the other macros are there. For example, the `%MktEx` macro calls: the `%MktRuns` macro to parse the factors list, the `%MktDes` macro for candidate set generation and search, and the `%MktOrth` macro to get the orthogonal array catalog.

[*]All of these macros come with the following disclaimer: This information is provided by SAS institute Inc. as a service to its users. It is provided "as is". There are no warranties, expressed or implied, as to merchantability or fitness for a particular purpose regarding the accuracy of the materials or code contained herein.

The macros do not have to be included (for example, with a `%include` statement). They can be called directly once they are properly installed. For more information about autocall libraries, see *SAS Macro Language: Reference*. On a PC for example, the autocall library might be installed in the `stat\sasmacro` directory off of your SAS root directory. The name of your SAS root directory could be anything, but it is probably something like C:\Program Files\SAS\SASFoundation\9.2. One way to find the right directory is to use `Start` → `Find` to find one of the macros such as `mktex.sas`. The following are some examples of SAS/STAT autocall macro directories on a PC:

```
C:\Program Files\SAS\SASFoundation\9.2\stat\sasmacro
C:\Program Files\SAS\SAS 9.1\stat\sasmacro
```

Unix has a similar directory structure to the PC. The autocall library in Unix might be installed in the `stat/sasmacro` directory off of your SAS root directory. On MVS, each macro is a different member of a PDS. For details on installing autocall macros, consult your host documentation.

Usually, an autocall library is a directory containing individual files, each of which contains one macro definition. An autocall library can also be a SAS catalog. To use a directory as a SAS autocall library, store the source code for each macro in a separate file in the directory. The name of the file must be the same as the macro name, typically followed by `.sas`. For example, the macro `%MktEx` must typically be stored in a file named `mktex.sas`. On most hosts, the reserved `fileref sasautos` is assigned at invocation time to the autocall library supplied by SAS or another one designated by your site.

The libraries that SAS searches for autocall macros are controlled by the SASAUTOS option. The default value for this option is set in the configuration file. One way to have SAS find your autocall macros is to update this option in the SAS configuration file. For example, searching for SAS*.CFG might turn up SASV9.CFG on a PC that will contain lines like the following:

```
/* Setup the SAS System autocall library definition */
-SET SASAUTOS  (
                "!sasroot\core\sasmacro"
                "!sasext0\ets\sasmacro"
                "!sasext0\graph\sasmacro"
                "!sasext0\iml\sasmacro"
                "!sasext0\qc\sasmacro"
                "!sasext0\share\sasmacro"
                "!sasext0\stat\sasmacro"
               )
```

The directories listed in this option will depend on what SAS products you have licensed. If you replace the existing macros in `stat\sasmacro`, you should never have to change this file. However, if you install the new macros anywhere else, you need to ensure that that location appears in your configuration file *before* `stat\sasmacro` or you will not get all of the most recent macros. For details, see your host documentation and SAS macro language documentation.

%ChoicEff Macro

The `%ChoicEff` autocall macro finds efficient experimental designs for choice experiments and evaluates choice designs. You supply a set of candidates. The macro searches the candidates for an efficient experimental design—a design in which the variances of the parameter estimates are minimized, given an assumed parameter vector $\boldsymbol{\beta}$.

If you are anxious to get started on choice designs, and you want to start immediately, then this is the right place. The examples in the examples section illustrate all of the tools that you need to make good choice designs. They do not illustrate all of the tools that you can use or always illustrate the very best methods for your particular situation, but they illustrate the minimum subset of tools you need to do virtually anything you might ever need to do in the area of choice design. If instead, you want to begin by better understanding what you are doing, be sure to read the experimental design chapter beginning on page 53.

You should also see the discrete choice chapter starting on page 285. Note though that the discrete choice chapter is an older chapter, and the approach that it emphasizes (making a choice design through the `%MktEx` and `%MktRoll` macros) is not in use as much as it once was. While that approach is important, and it can in fact create optimal designs for some problems, you can make all of the designs that you need with the more limited tool set described in this chapter where the `%MktEx` macro only makes candidate sets for the `%ChoicEff` macro. When you become a sophisticated designer of choice experiments, you will want to be facile with all of the tools in the discrete choice chapter, the experimental design chapter, and this chapter. However, when you are just starting, you might find it easer to concentrate on simply using the `%ChoicEff` macro with a candidate set of alternatives that you create by using the `%MktEx` macro.

See the following pages for examples of using this macro in the design chapter: 81, 83, 85, 87, 109, 112, 137, 140, 142, 170, 193 and 203. Also see the following pages for examples of using this macro in the discrete choice chapter: 313, 317, 320, 322, 360, 365, 366, 430, 508, 509, 542, 559, 564, 567, 570, 570, 574, 576, 597, 599, 607, 618, 628, 632, 636, 645, 650, 654, 656, 659 and 662. Additional examples appear throughout this chapter.

There are four ways that you can use the `%ChoicEff` macro:

- You create a candidate set of alternatives, and the macro creates a design consisting of choice sets built from the alternatives you supplied. You must designate for each candidate alternative the design alternative(s) for which it is a candidate. For a generic design, you create one list of candidate alternatives, and each candidate can be used for every alternative in the design. For a branded study with m brands, you must create a list with m types of candidate alternatives, one for each brand.

- You create a candidate set of choice sets, and the macro builds a design from the choice sets that you supplied. This approach was designed to handle restrictions across alternatives (certain alternatives may not appear with certain other alternatives) and with partial-profile designs (Chrzan and Elrod 1995). However, the candidate set of alternatives approach along with the new `restrictions=` option (described next) is often better than this approach. This is because for all but the smallest designs, the candidate set of choice set approach considers much smaller subsets of possible designs. Unless it is much easier for you to create a candidate set of restricted choice sets than to create a restrictions macro, you should use the `restrictions=` option and a candidate set of choice sets instead of a candidate set of choice sets.

- You create a candidate set of alternatives and a macro that provides restrictions on how the alternatives can be used to make the design. The `%ChoicEff` macro creates a design consisting of choice sets built from the alternatives you supplied. You must designate for each candidate alternative the design alternative(s) for which it is a candidate. For a generic design, you create one list of candidate alternatives, and each candidate can be used for every alternative in the design. For a branded study with m brands, you must create a list with m types of candidate alternatives, one for each brand. You can restrict the design in any way (within alternatives, across alternatives and within choice sets, or even across choice sets). For example, you can use the restrictions macro to prevent dominated alternatives, to force or prevent overlap in factor levels within choice set, to prevent certain levels from occurring with other levels, to force constant attributes within choice set, to control the number of constant attributes across choice sets, and so on.
- You supply a choice design, and the `%ChoicEff` macro evaluates it. The choice design might have been created by a previous run of the `%ChoicEff` macro, or by the `%MktEx` macro, or by other means.

The `%ChoicEff` macro uses a modified Fedorov candidate-set-search algorithm, just like the OPTEX procedure and the parts of the `%MktEx` macro. Typically, you use as a candidate set a full-factorial design, a fractional-factorial design, or an orthogonal array created with the `%MktEx` macro. First, the `%ChoicEff` macro either constructs a random initial design from the candidates or it uses an initial design that you specified. The macro considers swapping out every design alternative/set and replacing it with each candidate alternative/set. Swaps that increase efficiency are performed. The process of evaluating and swapping continues until efficiency stabilizes at a local maximum. This process is repeated with different initial designs, and the best design is output for use. The key differences between the `%ChoicEff` macro and the `%MktEx` macro are as follows:

- The `%ChoicEff` macro requires you to specify the true (or assumed true) parameters and it optimizes the variance matrix for a multinomial logit discrete choice model, which is a nonlinear model.
- The `%MktEx` macro optimizes the variance matrix for a linear model, which does not depend on the parameters.

Examples

The example section provides a series of examples of different ways that you can use the `%ChoicEff` macro.

Generic Choice Design Constructed from Candidate Alternatives

This example creates a design for a generic choice model with 3 three-level factors. First, you use the `%MktEx` macro to create a set of candidate alternatives, where `x1-x3` are the factors. Note that the `n=` specification accepts expressions. The following steps make and display the candidate set:

```
%mktex(3 ** 3, n=3**3, seed=238)

proc print; run;
```

The results are as follows:

```
Obs    x1    x2    x3

  1     1     1     1
  2     1     1     2
  3     1     1     3
  4     1     2     1
  5     1     2     2
  6     1     2     3
  7     1     3     1
  8     1     3     2
  9     1     3     3
 10     2     1     1
 11     2     1     2
 12     2     1     3
 13     2     2     1
 14     2     2     2
 15     2     2     3
 16     2     3     1
 17     2     3     2
 18     2     3     3
 19     3     1     1
 20     3     1     2
 21     3     1     3
 22     3     2     1
 23     3     2     2
 24     3     2     3
 25     3     3     1
 26     3     3     2
 27     3     3     3
```

Next, you run the `%ChoicEff` macro to find an efficient design for the unbranded, purely generic choice model assuming $\boldsymbol{\beta} = \mathbf{0}$ as follows:[*]

```
%choiceff(data=design,              /* candidate set of alternatives        */
          model=class(x1-x3 / sta), /* model with stdz orthogonal coding    */
          nsets=9,                  /* number of choice sets                */
          flags=3,                  /* 3 alternatives, generic candidates   */
          seed=289,                 /* random number seed                   */
          maxiter=60,               /* maximum number of designs to make    */
          options=relative,         /* display relative D-efficiency        */
          beta=zero)                /* assumed beta vector, Ho: b=0         */

proc print; var x1-x3; id set; by set; run;

proc print data=bestcov label;
   title 'Variance-Covariance Matrix';
   id __label;
   label __label = '00'x;
   var x:;
   run;
title;

%mktdups(generic, data=best, factors=x1-x3, nalts=3)
```

The option `data=final` names the input data set, `model=class(x1-x3 / sta)` specifies the PROC TRANSREG `model` statement for coding the design (the `sta` option, short for `standorth`, uses a standardize orthogonal coding), `nsets=9` specifies nine choice sets, `options=relative` requests that relative *D*-efficiency be displayed, `flags=3` specifies that there are three alternatives in a purely generic design,[†] `beta=zero` specifies all zero parameters, and `seed=382` specifies the random number seed, and `maxiter=50` specifies the number of designs to create. The design and the covariance matrix is displayed, and the design is checked for duplicates. Some of the results are as follows:

```
                         n    Name    Beta    Label

                         1    x11       0     x1 1
                         2    x12       0     x1 2
                         3    x21       0     x2 1
                         4    x22       0     x2 2
                         5    x31       0     x3 1
                         6    x32       0     x3 2
                   .
                   .
                   .
```

[*]If you are not running version 9.2 or a later SAS release, remove the slash and the `sta` option from the `model=` specification. The standardized orthogonal contrast coding is first available with Version 9.2 of SAS. Without this option, you will not get a relative *D*-efficiency in the 0 to 100 range.

[†]The option name `flags=` comes from the fact that in the context of other types of designs (designs with brands or labeled alternatives), this option provides a set of "flag" variables that specify which candidates can be be used for which alternatives.

```
      Design   Iteration  D-Efficiency       D-Error
      ----------------------------------------------
          52       0           4.74648       0.21068
                   1           9.00000 *     0.11111
                   2           9.00000       0.11111

                    .
                    .
                    .

                           Final Results

                     Design                 52
                     Choice Sets             9
                     Alternatives            3
                     Parameters              6
                     Maximum Parameters     18
                     D-Efficiency       9.0000
                     Relative D-Eff   100.0000
                     D-Error            0.1111
                     1 / Choice Sets    0.1111

        Variable                                       Standard
   n      Name      Label    Variance    DF      Error

   1      x11       x1 1      0.11111     1      0.33333
   2      x12       x1 2      0.11111     1      0.33333
   3      x21       x2 1      0.11111     1      0.33333
   4      x22       x2 2      0.11111     1      0.33333
   5      x31       x3 1      0.11111     1      0.33333
   6      x32       x3 2      0.11111     1      0.33333
                                         ==
                                          6

                       Set    x1    x2    x3

                        1      3     1     3
                               1     3     1
                               2     2     2

                        2      2     2     3
                               1     3     2
                               3     1     1

                        3      3     2     2
                               2     1     1
                               1     3     3
```

```
                                 4         2         1         3
                                           1         2         2
                                           3         3         1

                                 5         1         2         1
                                           2         1         2
                                           3         3         3

                                 6         3         1         2
                                           1         2         3
                                           2         3         1

                                 7         3         2         1
                                           2         3         3
                                           1         1         2

                                 8         1         1         3
                                           3         3         2
                                           2         2         1

                                 9         1         1         1
                                           3         2         3
                                           2         3         2


                              Variance-Covariance Matrix

               x1 1         x1 2         x2 1         x2 2         x3 1        x3 2

     x1 1     0.11111      0.00000     -0.00000      0.00000      0.00000      0.00000
     x1 2     0.00000      0.11111      0.00000     -0.00000      0.00000     -0.00000
     x2 1    -0.00000      0.00000      0.11111      0.00000      0.00000      0.00000
     x2 2     0.00000     -0.00000      0.00000      0.11111      0.00000     -0.00000
     x3 1     0.00000      0.00000      0.00000      0.00000      0.11111      0.00000
     x3 2     0.00000     -0.00000      0.00000     -0.00000      0.00000      0.11111
```

The output from the `%ChoicEff` macro consists of a list of the parameter names, values and labels, followed by a series of iteration histories (each based on a different random initial design), followed by a brief report on the most efficient design found, and finally a table with the parameter names, variances, *df*, and standard errors. The design and covariance matrix are displayed using PROC PRINT.

This design is optimal; it has 100% relative *D*-efficiency. Because a generic (main-effects only) design is requested with $\boldsymbol{\beta} = \mathbf{0}$, and the standardized orthogonal contrast coding is used, it is possible to get a relative *D*-efficiency on a 0 to 100 scale. For many other models, this will not be the case. Relative *D*-efficiency is *D*-efficiency divided by the number of choice sets, then multiplied by 100. In an optimal generic design such as this one, all of the following hold:

- *D*-efficiency equals the number of choice sets.
- *D*-error equals one over the number of choice sets.
- All of the variances equal one over the number of choice sets.

- All of the covariances are zero.
- Relative *D*-efficiency equals 100.

Note that *D*-error is by definition equal to the inverse of *D*-efficiency. Also note that in practice, the computed values of the covariances are often not precisely zero because inexact floating point arithmetic is involved. This is why some display as –0.00000 in the output.

The `%MktDups` macro reports the following:

```
Design:          Generic
Factors:         x1-x3
                 x1 x2 x3
Sets w Dup Alts: 0
Duplicate Sets:  0
```

There are no duplicate choice sets or duplicate alternatives within choice sets.

You can assign more descriptive names to the factors and values for the levels, for example, as follows:

```
proc format;
   value sf 1 = '12 oz' 2 = '16 oz' 3 = '24 oz';
   value cf 1 = 'Red  ' 2 = 'Green' 3 = 'Blue ';
   run;

data ChoiceDesign;
   set best;
   format x1 sf. x2 cf. x3 dollar6.2;
   label x1 = 'Size' x2 = 'Color' x3 = 'Price';
   x3 + 0.49;
   run;

proc print label; var x1-x3; id set; by set; run;
```

The final design is as follows:

```
                        Set    Size     Color     Price

                         1     24 oz    Red       $3.49
                               12 oz    Blue      $1.49
                               16 oz    Green     $2.49

                         2     16 oz    Green     $3.49
                               12 oz    Blue      $2.49
                               24 oz    Red       $1.49

                         3     24 oz    Green     $2.49
                               16 oz    Red       $1.49
                               12 oz    Blue      $3.49
```

```
                                4     16 oz    Red       $3.49
                                      12 oz    Green     $2.49
                                      24 oz    Blue      $1.49

                                5     12 oz    Green     $1.49
                                      16 oz    Red       $2.49
                                      24 oz    Blue      $3.49

                                6     24 oz    Red       $2.49
                                      12 oz    Green     $3.49
                                      16 oz    Blue      $1.49

                                7     24 oz    Green     $1.49
                                      16 oz    Blue      $3.49
                                      12 oz    Red       $2.49

                                8     12 oz    Red       $3.49
                                      24 oz    Blue      $2.49
                                      16 oz    Green     $1.49

                                9     12 oz    Red       $1.49
                                      24 oz    Green     $3.49
                                      16 oz    Blue      $2.49
```

Flag Variables

This example uses the `flags=3` option in the `%ChoicEff` macro as follows:

```
   %choiceff(data=design,             /* candidate set of alternatives        */
            model=class(x1-x3 / sta), /* model with stdz orthogonal coding    */
            nsets=9,                  /* number of choice sets                */
            flags=3,                  /* 3 alternatives, generic candidates   */
            seed=289,                 /* random number seed                   */
            maxiter=60,               /* maximum number of designs to make    */
            options=relative,         /* display relative D-efficiency        */
            beta=zero)                /* assumed beta vector, Ho: b=0         */
```

This is a short-hand notation for specifying flag variables. Your specifications must contain information that show or “flag” which candidate can be used in which alternative. In a generic design, you can create this information by creating flag variables yourself, or you can use the short-hand specification and have the `%ChoicEff` macro create these variables for you. When you create the flag variables yourself, you create one variable for each alternative. (With three alternatives, you create three flag variables.) The flag variables flag which candidates can be used for which alternative(s). Since this is a generic choice model, each candidate can appear in any alternative, which means you need to add flags that are constant and all one: `f1=1 f2=1 f3=1`. You can use the `%MktLab` macro to add the flag variables, essentially by specifying that you have three intercepts, you can program it yourself with a DATA step, or you can let the `%ChoicEff` macro create the flag variables for you. The option `int=f1-f3` in the `%MktLab` macro creates three variables with values that are all one. A 1 in variable `f1` indicates that the candidate can be used for the first alternative, a 1 in `f2` indicates that the candidate can be used for the second alternative, and so on. In this case, all candidates can be used for all alternatives, otherwise the flag variables contain zeros for candidates that cannot be used for certain alternatives. The default output data set from the `%MktLab` macro is called FINAL and is specified in the `data=` option in the `%ChoicEff` macro. The following step illustrates:

```
%mktlab(data=design, int=f1-f3)

proc print data=final; run;

%choiceff(data=final,                /* candidate set of alternatives        */
          model=class(x1-x3 / sta), /* model with stdz orthogonal coding    */
          nsets=9,                  /* number of choice sets                */
          flags=f1-f3,              /* flag which alt can go where, 3 alts  */
          seed=289,                 /* random number seed                   */
          maxiter=60,               /* maximum number of designs to make    */
          options=relative,         /* display relative D-efficiency        */
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

The candidates along with the flag variables are as follows:

```
                         Obs    f1    f2    f3    x1    x2    x3

                           1     1     1     1     1     1     1
                           2     1     1     1     1     1     2
                           3     1     1     1     1     1     3
                           4     1     1     1     1     2     1
                           5     1     1     1     1     2     2
                           6     1     1     1     1     2     3
                           7     1     1     1     1     3     1
                           8     1     1     1     1     3     2
                           9     1     1     1     1     3     3
                          10     1     1     1     2     1     1
                          11     1     1     1     2     1     2
                          12     1     1     1     2     1     3
                          13     1     1     1     2     2     1
                          14     1     1     1     2     2     2
                          15     1     1     1     2     2     3
                          16     1     1     1     2     3     1
                          17     1     1     1     2     3     2
                          18     1     1     1     2     3     3
```

```
        19       1       1       1       3       1       1
        20       1       1       1       3       1       2
        21       1       1       1       3       1       3
        22       1       1       1       3       2       1
        23       1       1       1       3       2       2
        24       1       1       1       3       2       3
        25       1       1       1       3       3       1
        26       1       1       1       3       3       2
        27       1       1       1       3       3       3
```

The results of the `%ChoicEff` macro match the results from the previous part of the example.

Direct Construction of an Optimal Generic Choice Design

These next steps directly create an optimal design for this generic choice model and evaluate its efficiency using the `%ChoicEff` macro and the initial design options. This generic design is created with only 3 choice sets this time, and it is constructed as follows:

```
%mktex(3 ** 4, n=9)

%mktlab(data=design, vars=Set Size Color Price)

proc print data=final; id set; by set; var size -- price; run;
```

The design is as follows:

```
Set    Size    Color    Price

 1       1       1        1
         2       3        2
         3       2        3

 2       1       3        3
         2       2        1
         3       1        2

 3       1       2        2
         2       1        3
         3       3        1
```

Notice that each attribute contains all three levels in each choice set.

The following steps evaluate the design:

```
%choiceff(data=final,               /* candidate set of choice sets         */
          init=final(keep=set),     /* select these sets from candidates    */
          intiter=0,                /* evaluate without internal iterations */
          model=class(size color    /* main-effects model with stdz         */
                      price / sta), /* orthogonal coding                    */
          nsets=3,                  /* number of choice sets                */
          nalts=3,                  /* number of alternatives               */
          options=relative,         /* display relative D-efficiency        */
          beta=zero)                /* assumed beta vector, Ho: b=0         */

%mktdups(generic, data=best, factors=size color price, nalts=3)
```

When we evaluate a design, we need to provide the design in the `data=` specification. Usually, you use the `data=` option to specify the candidate set to be searched. In some sense, the `data=` design is a candidate set in this context as well, and we use the `init=` option to specify how the initial design is constructed from the candidate set. The initial design is constructed by selecting the candidates specified in the `Set` variable in the `init=` data set. This is accomplished with the `init=final(keep=set)` specification. Then the `%ChoiceEff` macro selects just the specified candidate choice sets (in this case all of them) and uses them as the initial design. Specify `intiter=0` (internal iterations equals zero) when you just want to evaluate the efficiency of a given design and not improve on it. The output from the `%ChoiceEff` macro is as follows:

```
                    n    Name    Beta    Label

                    1    x11       0     x1 1
                    2    x12       0     x1 2
                    3    x21       0     x2 1
                    4    x22       0     x2 2
                    5    x31       0     x3 1
                    6    x32       0     x3 2

          Design   Iteration  D-Efficiency      D-Error
          ----------------------------------------------
               1       0           3.00000 *     0.33333


                          Final Results

                    Design                  1
                    Choice Sets             3
                    Alternatives            3
                    Parameters              6
                    Maximum Parameters      6
                    D-Efficiency       3.0000
                    Relative D-Eff   100.0000
                    D-Error            0.3333
                    1 / Choice Sets    0.3333
```

```
            Variable                                                Standard
     n        Name        Label      Variance      DF          Error

     1        x11         x1 1        0.33333       1        0.57735
     2        x12         x1 2        0.33333       1        0.57735
     3        x21         x2 1        0.33333       1        0.57735
     4        x22         x2 2        0.33333       1        0.57735
     5        x31         x3 1        0.33333       1        0.57735
     6        x32         x3 2        0.33333       1        0.57735
                                                   ==
                                                    6
```

This design is optimal. Relative *D*-efficiency is 100%, and all of the variances of the parameter estimates are equal to one over the number of choice sets.

The `%MktDups` macro reports the following:

```
Design:           Generic
Factors:          size color price
                  Color Price Size
Sets w Dup Alts: 0
Duplicate Sets:  0
```

There are no duplicate choice sets or duplicate alternatives within choice sets.

The following steps create and evaluate an equivalent but slightly different version of the optimal generic choice design:

```
%mktex(3 ** 4, n=9, options=nosort)

proc sort; by x4 x1; run;

%mktlab(data=design, vars=Size Color Price Set)

proc print; id set; by set; run;

%choiceff(data=final,               /* candidate set of choice sets         */
          init=final(keep=set),     /* select these sets from candidates    */
          model=class(size color    /* main-effects model with stdz         */
                      price / sta), /* orthogonal coding                    */
          nsets=3,                  /* number of choice sets                */
          nalts=3,                  /* number of alternatives               */
          options=relative,         /* display relative D-efficiency        */
          beta=zero)                /* assumed beta vector, Ho: b=0         */

%mktdups(generic, data=best, factors=size color price, nalts=3)
```

The full results are not shown, but the design is as follows:

```
                    Set    Size    Color    Price

                     1       1       1        1
                             2       2        2
                             3       3        3


                     2       1       2        3
                             2       3        1
                             3       1        2


                     3       1       3        2
                             2       1        3
                             3       2        1
```

It has a cyclic structure where the second and third alternatives are constructed from the previous alternative by adding 1 (mod 3).[*] The levels for just the first alternative for each set are as follows:

```
                             1      1      1
                             1      2      3
                             1      3      2
```

This matrix is known as a 3×3 difference scheme of order 3.[†] While we will not prove any of this here or anywhere else in this book, the following fact is used in a number of places in this book. The cyclic development of a difference scheme (that is, making subsequent p-level alternative from previous alternatives by adding 1 mod p), is an algorithm for making optimal generic choice designs. See the section beginning on 102 for more information about optimal generic choice designs.

[*]More precisely, since these numbers are based on one instead of zero, the operation is: (x mod 3) + 1.

[†]Although more typically, we think of this matrix minus one as the difference scheme.

Generic Choice Design Constructed from Candidate Choice Sets

This example is provided for complete coverage of the `%ChoicEff` macro. If you are just getting started, concentrate instead on examples of the `%ChoicEff` macro that use candidate sets of alternatives. The next example starts on page 820.

These next steps use the `%MktEx` and `%MktRoll` macros to create a candidate set of choice sets and the `%ChoicEff` macro to search for an efficient design using the candidate-set-swapping algorithm:

```
   %mktex(3 ** 9, n=2187, seed=368)

   %mktroll(design=design, key=3 3, out=rolled)

   %choiceff(data=rolled,              /* candidate set of choice sets         */
             model=class(x1-x3 / sta), /* model with stdz orthogonal coding    */
             nsets=9,                  /* number of choice sets                */
             nalts=3,                  /* number of alternatives               */
             maxiter=20,               /* maximum number of designs to make    */
             seed=205,                 /* random number seed                   */
             options=relative nodups,  /* display relative D-eff, avoid dups   */
             beta=zero)                /* assumed beta vector, Ho: b=0         */

   %mktdups(generic, data=best, factors=x1-x3, nalts=3)
```

The first steps create a candidate set of choice sets. The `%MktEx` macro creates a design with nine factors, three for each of the three alternatives. The `%MktRoll` macro with `key=3 3` makes the following `Key` data set:

```
                              x1    x2    x3

                              x1    x2    x3
                              x4    x5    x6
                              x7    x8    x9
```

It specifies that the first alternative is made from the linear arrangement factors `x1-x3`, the second alternative is made from `x4-x6`, and the third alternative is made from `x7-x9`. The `%MktRoll` macro turns a linear arrangement of a choice design into a true choice design using the rules specified in the `Key` data set.

In the `%ChoicEff` macro, the `nalts=3` option specifies that there are three alternatives. There must always be a constant number of alternatives in each choice set, even if all of the alternatives will not be used. When a nonconstant number of alternatives is desired, you must use a weight variable to flag those alternatives that the subject will not see (see for example page 912). When you swap choice sets, you need to specify `nalts=`. The output from these steps is not appreciably different from what we saw previously, so it is not shown. The `%ChoicEff` macro can on occasion find a 100% *D*-efficient generic choice design with this approach. However, the optimal design is much harder to find when using a large candidate set of choice sets than when using a small candidate set of alternatives. This is one reason why the candidate set of alternatives approach (potentially with restrictions) is usually preferred over creating a candidate set of choice sets.

Avoiding Dominated Alternatives

In this next example, there are 6 four-level attributes, which are all quantitative in nature. As the factor level value increases, the desirability of the feature increases. Of course, dominance could go in the other direction (the desirability increases as the level decreases as in price), and that can easily be handled as well. When one alternative contains levels that are all less than or equal to the levels for another alternative, the first alternative is dominated by the second. When one alternative is dominated by another, the choice task becomes easier for respondents. Eliminating dominated alternatives forces the respondents to consider all of the attributes and all of the alternatives in making a choice. The goal in this example is to write a restrictions macro that prevents dominated alternatives from occurring. The first step makes a candidate set of alternatives:

```
%mktex(4 ** 6, n=32, seed=104)
```

The next steps find a choice design where no alternatives dominate:

```
%macro res;
   do i = 1 to nalts;
      do k = i + 1 to nalts;
         if all(x[i,] >= x[k,])     /* alt i dominates alt k              */
            then bad = bad + 1;
         if all(x[k,] >= x[i,])     /* alt k dominates alt i              */
            then bad = bad + 1;
         end;
      end;
   %mend;

%choiceff(data=randomized,          /* candidate set of choice sets       */
          model=class(x1-x6 / sta), /* model with stdz orthogonal coding  */
          nsets=8,                  /* number of choice sets              */
          flags=4,                  /* 4 alternatives, generic candidates */
          seed=104,                 /* random number seed                 */
          options=relative          /* display relative D-efficiency      */
                  resrep,           /* detailed report on restrictions    */
          restrictions=res,         /* name of the restrictions macro     */
          resvars=x1-x6,            /* vars used in defining restrictions */
          maxiter=1,                /* maximum number of designs to make  */
          beta=zero)                /* assumed beta vector, Ho: b=0       */
```

First, a macro is created that counts dominated alternatives. The macro is written in IML. An IML scalar `bad` is increased by one every time a dominated alternative is found. Note that the scalar `bad` is automatically initialized to zero. In this example, the restrictions macro is evaluating each choice set at the time that the `%ChoicEff` macro is constructing it. The current choice set that is being considered is stored in the matrix `x`. When all elements in the *ith* row of `x` are greater than or equal to all elements in the *kth* row of `x`, the *ith* row dominates the *kth* row and `bad` is increased by one. Similarly, when all elements in the *kth* row of `x` are greater than or equal to all elements in the *ith* row of `x`, the *kth* row dominates the *ith* row and `bad` is increased by one. The `do` loops set `i` and `k` to (1,2), (1,3), (1,4) (2,3), (2,4), and (3,4), which are all pairs of attributes within a choice set.

The expression `all(x[i,] >= x[k,])` works as follows. The expression `(x[i,] >= x[k,])` compares two row vectors, the *ith* row in `x` and the *kth* row in `x`. The result of comparing two vectors with a Boolean operation (in this case, greater than or equal to) is another vector, of the same order as the two vectors being compared, that consists of ones when the element-wise comparison is true and zeros when the element-wise comparisons are false. The `all` function returns a 1 or true when all elements in the scalar, vector, or matrix argument are nonzero and nonmissing. Otherwise if there are any zeros or missings, it returns a zero or false. For example, `all({1 2 3} >= {3 2 1})` = `all({0 1 1})` = 0 (or false) and `all({3 2 4} >= {1 2 3})` = `all({1 1 1})` = 1 (or true).

The `%ChoicEff` macro uses the `res` macro when it considers swapping alternatives into the design. You must specify two options when there are restrictions. The `restrictions=`*macro-name* option provides the name of the macro that evaluates the badness of each choice set. The `resvars=`*variable-list* option provides the names of the variables that are in the design. The option `maxiter=1` is used to make only one design for now during the testing phase until you are sure that you are specifying restrictions correctly. The option `options=resrep` adds additional output to the iteration history to show how the macro is progressing in producing a design that conforms to the restrictions.

Some of the results are as follows:

```
                 Design   Iteration  D-Efficiency      D-Error
                 ----------------------------------------------
                      1       0          2.69601 *     0.37092
        at    1   1 swapped in     1       2.91185 bad =       1
        at    1   2 swapped in     2       3.04604 bad =       0
        at    1   3 swapped in    11       3.14498 bad =       0
        at    1   4 swapped in     9       3.22393 bad =       0
        at    1   4 swapped in    20       3.28990 bad =       0
        at    1   4 swapped in    27       3.34674 bad =       0
        at    2   1 swapped in     1       3.54869 bad =       2
        at    2   1 swapped in     2       3.54869 bad =       0
        at    2   1 swapped in     3       3.57907 bad =       0
        at    2   1 swapped in     5       3.69173 bad =       0
        at    2   1 swapped in    31       3.72044 bad =       0
        at    2   2 swapped in     2       3.92700 bad =       0
        at    2   2 swapped in     4       3.93246 bad =       0
        at    2   2 swapped in     5       4.05283 bad =       0
        at    2   2 swapped in    23       4.06014 bad =       0
        at    2   3 swapped in     3       4.08614 bad =       0
        at    2   3 swapped in     5       4.17255 bad =       0
        at    2   3 swapped in     8       4.18298 bad =       0
        at    2   3 swapped in    20       4.26063 bad =       0
        at    2   4 swapped in    29       4.26063 bad =       0
        at    3   1 swapped in     1       4.42830 bad =       2
        at    3   1 swapped in     2       4.42830 bad =       0
        at    3   1 swapped in     3       4.43255 bad =       0
        at    3   1 swapped in     4       4.45001 bad =       0
        at    3   1 swapped in     5       4.63752 bad =       0
        at    3   2 swapped in     3       4.67839 bad =       0
```

```
at    3   2 swapped in     4       4.69266 bad =       0
at    3   2 swapped in     6       4.77758 bad =       0
at    3   3 swapped in     1       4.82692 bad =       0
at    3   3 swapped in     4       4.91595 bad =       0
at    3   4 swapped in     7       4.96346 bad =       0
at    3   4 swapped in    12       5.06320 bad =       0
.
.
.
at    7   4 swapped in    17       5.76862 bad =       0
at    8   1 swapped in     1       5.83926 bad =       0
at    8   1 swapped in     3       5.91170 bad =       0
at    8   2 swapped in    30       5.91170 bad =       0
at    8   3 swapped in    24       5.94511 bad =       0
at    8   4 swapped in    22       5.94511 bad =       0
                       1           5.94511 *     0.16821
.
.
.
at    1   1 swapped in     1       6.39883 bad =       0
at    1   2 swapped in     2       6.39883 bad =       0
at    1   3 swapped in    20       6.39883 bad =       0
at    1   4 swapped in    16       6.39883 bad =       0
at    3   1 swapped in     5       6.39883 bad =       0
at    3   2 swapped in     9       6.39883 bad =       0
at    3   3 swapped in     4       6.39883 bad =       0
at    3   3 swapped in     7       6.46427 bad =       0
at    3   4 swapped in    12       6.46427 bad =       0
.
.
.
at    8   1 swapped in     3       6.54545 bad =       0
at    8   2 swapped in    30       6.54545 bad =       0
at    8   3 swapped in    24       6.54545 bad =       0
at    8   4 swapped in    22       6.54545 bad =       0
                       3           6.54545 *     0.15278
at    1   1 swapped in     1       6.54545 bad =       0
at    1   2 swapped in     2       6.54545 bad =       0
at    1   3 swapped in    20       6.54545 bad =       0
at    1   4 swapped in    16       6.54545 bad =       0
                       4           6.54545       0.15278
```

First, an ordinary line is printed in the iteration history table, which displays the efficiency of the initial random design. Next, there is a line that provides information about every swap that is performed. In choice set *i*, alternative *j*, candidate alternative *k* was swapped in resulting in a *D*-efficiency of *a* and a new value for `bad` of *b*. You can see that badness does not always start out at zero but it quickly goes to zero. An ordinary line is written to the iteration history table whenever a full pass through the design is completed.

Lines beginning with “`at`” are added by `options=resrep`. All other lines appear by default. The first line (`1 0 2.69601 0.37092`) provides the design number (1), iteration number (0), *D*-efficiency (2.69601), and *D*-error (0.37092) for the initial random design. The next line (`at 1 1 swapped in 1 2.91185 bad = 1`) specifies that in choice set 1 and alternative 1, candidate alternative 1 is swapped in resulting in a *D*-efficiency of 2.911854 and a badness value of 1. The initial badness is not stated, but it must have been greater than zero. No other candidate alternatives improve the badness or efficiency for alternative 1 of set 1, so no other swaps are performed. However, the next line (`at 1 2 swapped in 2 3.04604 bad = 0`) shows that in alternative 2 of set 1, candidate alternative 2 is swapped in and badness is reduced to zero for this choice set. Recall that with this restrictions macro, badness is only evaluated within the current choice set, so a badness of zero at this point does not mean that the design conforms to all restrictions. This can be seen in the first swap for choice set 2 (`at 2 1 swapped in 1 3.54869 bad = 2`) where the badness for the second choice set is reduced to 2 for the first swap. Subsequent swaps reduce the badness to zero.

Iterations progress through the 8 choice sets until the line (`1 5.945110 0.168205`) shows that at the end of the first iteration (the end of the first full pass through the choice design) the *D*-efficiency is 5.94511 and the *D*-error is 0.16821. For this problem, badness is never more than zero in the second and subsequent iterations, although in general there are no guarantees that all violations will be fixed in the first iteration. The final line of the iteration history (`4 6.54545 0.15278`) displays the final *D*-efficiency and *D*-error. Notice that the number of swaps decreases for each iteration. This is usually the case. If `maxiter=1` had not been specified, this process is repeated with a different initial design selected at random from the candidates.

The design summary and the final efficiency results are as follows:

```
                              Final Results

                         Design                  1
                         Choice Sets             8
                         Alternatives            4
                         Parameters             18
                         Maximum Parameters     24
                         D-Efficiency       6.5455
                         Relative D-Eff    81.8181
                         D-Error            0.1528
                         1 / Choice Sets    0.1250
```

```
          Variable                                      Standard
    n       Name       Label     Variance     DF         Error

    1       x11        x1 1       0.13396      1       0.36601
    2       x12        x1 2       0.15708      1       0.39633
    3       x13        x1 3       0.19401      1       0.44046
    4       x21        x2 1       0.16605      1       0.40749
    5       x22        x2 2       0.13352      1       0.36540
    6       x23        x2 3       0.18423      1       0.42922
    7       x31        x3 1       0.24205      1       0.49199
    8       x32        x3 2       0.18793      1       0.43351
    9       x33        x3 3       0.14790      1       0.38458
   10       x41        x4 1       0.17957      1       0.42376
   11       x42        x4 2       0.19117      1       0.43723
   12       x43        x4 3       0.14291      1       0.37803
   13       x51        x5 1       0.18529      1       0.43045
   14       x52        x5 2       0.12508      1       0.35367
   15       x53        x5 3       0.15012      1       0.38746
   16       x61        x6 1       0.16184      1       0.40229
   17       x62        x6 2       0.14248      1       0.37747
   18       x63        x6 3       0.17398      1       0.41711
                                              ==
                                              18
```

This construction method creates a design that is 82% efficient relative to the optimal design with no restrictions. With only one iteration, we have no way of knowing how large the value might be with this candidate set. However, our concern at the moment is in evaluating our restrictions macro, not in finding the absolute maximum for *D*-efficiency.

The following steps assign names and levels for the attributes and display the design:

```
proc format;
   value x1f 1='Bad'   2='Good'    3='Better'   4='Best';
   value x2f 1='Small' 2='Average' 3='Bigger'   4='Large';
   value x3f 1='Ugly'  2='OK'      3='Average'  4='Nice ';
   value x4f 1='Slow'  2='Fast'    3='Faster'   4='Fastest';
   value x5f 1='Rough' 2='Normal'  3='Smoother' 4='Smoothest';
   value x6f 1='$9.99' 2='$8.99'   3='$7.99'    4='$6.99';
   run;

proc print label;
   label x1 = 'Quality'   x2 = 'Size'       x3 = 'Appearance'
         x4 = 'Speed'     x5 = 'Smoothness' x6 = 'Price';
   format x1 x1f. x2 x2f. x3 x3f. x4 x4f. x5 x5f. x6 x6f.;
   by set; id set; var x:;
   run;
```

Notice that levels are assigned so that in terms of the original values (1, 2, 3, 4), larger values are always better than smaller values. In particular, notice that the largest price is assigned to the smallest level (1 becomes $9.99) and the smallest price is assigned to the largest level (4 becomes $6.99).

The design is as follows:

```
Set    Quality     Size       Appearance    Speed      Smoothness    Price

 1     Good       Average      OK           Slow       Smoother      $9.99
       Best       Large        Ugly         Faster     Smoothest     $9.99
       Best       Small        OK           Slow       Rough         $8.99
       Bad        Large        Nice         Fastest    Normal        $7.99

 2     Best       Average      Nice         Fast       Normal        $9.99
       Better     Large        Average      Slow       Smoothest     $8.99
       Bad        Large        Ugly         Fast       Rough         $8.99
       Bad        Bigger       OK           Faster     Rough         $6.99

 3     Good       Small        Ugly         Fastest    Smoother      $7.99
       Best       Large        Nice         Slow       Smoother      $6.99
       Good       Bigger       Ugly         Faster     Normal        $8.99
       Better     Average      Average      Fast       Rough         $7.99

 4     Best       Small        Average      Faster     Normal        $7.99
       Best       Average      Ugly         Fastest    Rough         $6.99
       Bad        Small        OK           Fastest    Smoothest     $9.99
       Better     Bigger       Ugly         Fast       Smoother      $9.99

 5     Bad        Average      Ugly         Slow       Smoothest     $7.99
       Better     Large        OK           Faster     Smoother      $7.99
       Best       Bigger       Average      Fastest    Smoother      $8.99
       Good       Small        Nice         Fast       Smoothest     $8.99

 6     Good       Large        Average      Fastest    Rough         $9.99
       Good       Average      Average      Faster     Smoothest     $6.99
       Best       Bigger       OK           Fast       Smoothest     $7.99
       Better     Small        Nice         Faster     Rough         $9.99

 7     Good       Large        OK           Fast       Normal        $6.99
       Best       Large        Ugly         Faster     Smoothest     $9.99
       Better     Small        Ugly         Slow       Normal        $6.99
       Bad        Average      Nice         Faster     Smoother      $8.99

 8     Bad        Bigger       Average      Slow       Normal        $9.99
       Good       Bigger       Nice         Slow       Rough         $7.99
       Bad        Small        Average      Fast       Smoother      $6.99
       Better     Average      OK           Fastest    Normal        $8.99
```

The following steps provide a report and check on dominance:

```
title;
proc iml;
   use best(keep=x1-x6); read all into x;
   sets = 8;
   alts = 4;
   if sets # alts ^= nrow(x) then print 'ERROR: Invalid sets and/or alts.';
   do a = 1 to sets;
      print a[label='Set'] '   '
            (x[((a - 1) * alts + 1) : a * alts,])[format=1.] '          ';
      ii = 0;
      do i = (a - 1) * alts + 1 to a * alts;
         ii = ii + 1;
         kk = ii;
         do k = i + 1 to a * alts;
            kk = kk + 1;
            print ii[label='Alt'] '   ' (x[i,])[format=1.]
                  (sum(x[i,] >= x[k,]))[label='Sum'],
                  kk[label=none]  '   ' (x[k,])[format=1.]
                  (sum(x[k,] >= x[i,]))[label=none];
            if all(x[i,] >= x[k,]) | all(x[k,] >= x[i,]) then
               print "ERROR: Sum=0.";
            end;
         end;
      end;
   quit;
```

You should always double check your design to make sure you wrote your restrictions macro correctly. Some of the results are as follows:

```
                              Set

                                1     4 1 2 1 1 2
                                      2 1 1 4 3 3
                                      2 2 3 3 4 4
                                      1 3 3 1 2 1

                              Alt                       Sum

                                1     4 1 2 1 1 2         3
                                2     2 1 1 4 3 3         4

                              Alt                       Sum

                                1     4 1 2 1 1 2         1
                                3     2 2 3 3 4 4         5
```

```
                                      Alt                        Sum

                                        1     4 1 2 1 1 2          3
                                        4     1 3 3 1 2 1          4

                                      Alt                        Sum

                                        2     2 1 1 4 3 3          2
                                        3     2 2 3 3 4 4          5

                                      Alt                        Sum

                                        2     2 1 1 4 3 3          4
                                        4     1 3 3 1 2 1          2

                                      Alt                        Sum

                                        3     2 2 3 3 4 4          5
                                        4     1 3 3 1 2 1          2
```

The IML step displays each choice set, each pair of alternatives, and the number of attributes in which each alternative dominates the other. An error is printed if any alternative dominates another alternative in all attributes. For this design, no alternatives dominate.

Now consider for a moment what would happen if you made a mistake in your dominance evaluation macro and specified a set of restrictions that could not be satisfied:

```
%mktex(4 ** 6, n=32, seed=104)

%macro res;
   do i = 1 to nalts;
      do k = i + 1 to nalts;
         if any(x[i,] >= x[k,]) then bad = bad + 1; /* should be all not any */
         if any(x[k,] >= x[i,]) then bad = bad + 1; /* should be all not any */
         end;
      end;
   %mend;

%choiceff(data=randomized,          /* candidate set of choice sets         */
          model=class(x1-x6 / sta), /* model with stdz orthogonal coding    */
          nsets=8,                  /* number of choice sets                */
          flags=4,                  /* 4 alternatives, generic candidates   */
          seed=104,                 /* random number seed                   */
          options=relative          /* display relative D-efficiency        */
                  resrep,           /* detailed report on restrictions      */
          restrictions=res,         /* name of the restrictions macro       */
          resvars=x1-x6,            /* vars used in defining restrictions   */
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

Some of the results are as follows:

```
                    Design   Iteration  D-Efficiency      D-Error
                    ----------------------------------------------
                         1       0           2.69601 *     0.37092
          at    1   1 swapped in     1       2.91185 bad =      12
          at    1   2 swapped in     2       3.04604 bad =      12
          at    1   3 swapped in    11       3.14498 bad =      12
          at    1   4 swapped in     3       3.14498 bad =      12
          at    1   4 swapped in     9       3.22393 bad =      12
          at    1   4 swapped in    20       3.28990 bad =      12
          at    1   4 swapped in    27       3.34674 bad =      12
          at    2   1 swapped in     1       3.54869 bad =      12
          at    2   1 swapped in     2       3.54869 bad =      12
          at    2   1 swapped in     3       3.57907 bad =      12
          at    2   1 swapped in     5       3.69173 bad =      12
          at    2   1 swapped in    21       3.72136 bad =      12
          at    2   2 swapped in     1       3.90634 bad =      12
          at    2   2 swapped in     5       4.04435 bad =      12
          at    2   3 swapped in     1       4.04516 bad =      12
          at    2   3 swapped in     3       4.08971 bad =      12
          at    2   3 swapped in     4       4.09739 bad =      12
          at    2   3 swapped in     8       4.14098 bad =      12
          at    2   3 swapped in    12       4.26015 bad =      12
          at    2   3 swapped in    20       4.27800 bad =      12
          at    2   4 swapped in    29       4.27800 bad =      12
                .
                .
                .
                .


                                        Final Results

                              Design                  2
                              Choice Sets             8
                              Alternatives            4
                              Parameters             18
                              Maximum Parameters     24
                              D-Efficiency       5.9444
                              Relative D-Eff    74.3044
                              D-Error            0.1682
                              1 / Choice Sets    0.1250
                              WARNING: Restriction violations.
```

```
             Variable                                    Standard
        n      Name       Label     Variance    DF        Error

        1      x11        x1 1       0.21317     1       0.46170
        2      x12        x1 2       0.15990     1       0.39988
        3      x13        x1 3       0.25107     1       0.50107
        4      x21        x2 1       0.20685     1       0.45481
        5      x22        x2 2       0.20666     1       0.45460
        6      x23        x2 3       0.14918     1       0.38624
        7      x31        x3 1       0.21191     1       0.46033
        8      x32        x3 2       0.20148     1       0.44886
        9      x33        x3 3       0.18835     1       0.43399
       10      x41        x4 1       0.16921     1       0.41135
       11      x42        x4 2       0.20802     1       0.45610
       12      x43        x4 3       0.17490     1       0.41821
       13      x51        x5 1       0.16740     1       0.40914
       14      x52        x5 2       0.21489     1       0.46356
       15      x53        x5 3       0.15014     1       0.38748
       16      x61        x6 1       0.18323     1       0.42805
       17      x62        x6 2       0.19784     1       0.44479
       18      x63        x6 3       0.16916     1       0.41129
                                                ==
                                                18
```

It is clear from these results that this set of restrictions is impossible and is not met.

Forcing Attributes to be Constant

The following steps again find a restricted design, but with a different set of restrictions:

```
%mktex(4 ** 6, n=32, seed=104)

%macro res;
   c = 0;                         /* n of constant attrs in x            */
   do i = 1 to ncol(x);
      c = c +                     /* n of constant attrs in x            */
         all(x[,i] = round(x[:,i]));/* all values equal average value    */
      end;
   bad = bad + abs(c - 2);        /* want two attrs constant             */
   %mend;
```

```
%choiceff(data=randomized,          /* candidate set of alternatives        */
          model=class(x1-x6 / sta), /* model with stdz orthogonal coding    */
          nsets=8,                  /* number of choice sets                */
          flags=4,                  /* 4 alternatives, generic candidates   */
          seed=104,                 /* random number seed                   */
          options=relative          /* display relative D-efficiency        */
                  resrep,           /* detailed report on restrictions      */
          restrictions=res,         /* name of the restrictions macro       */
          resvars=x1-x6,            /* variable names used in restrictions  */
          maxiter=1,                /* maximum number of designs to make    */
          bestout=desres2,          /* final choice design                  */
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

In this case, a `do` statement loops over every column in the choice set. The IML scalar `c` is incremented by one every time all values in column *i* are equal to the average level value.[*] In other words, it counts the number of constant columns. The value of `bad` is the absolute difference between `c` and 2. The goal is to create a choice design where exactly two attributes are constant in each choice set.

Part of the iteration history is as follows:

```
                 Design   Iteration  D-Efficiency      D-Error
                 ----------------------------------------------
                     1       0           2.69601 *     0.37092
         at    1    1 swapped in     1       2.91185 bad =       2
         .
         .
         .
         at    1    2 swapped in     2       3.04604 bad =       2
         at    2    1 swapped in     1       3.32110 bad =       2
         .
         .
         .
         at    2    4 swapped in    29       3.98253 bad =       2
         at    3    1 swapped in     1       4.19037 bad =       2
         .
         .
         .
         at    3    4 swapped in     6       4.75032 bad =       2
         at    4    1 swapped in     1       4.80793 bad =       2
         .
         .
         .
         at    4    4 swapped in    13       5.05268 bad =       2
```

[*]Note that the expression `x[:,i]` extracts column *i* from matrix `x` then computes the mean over the rows of the selected column. The colon is a subscript reduction operator that computes the mean.

```
at    5   1 swapped in     1       5.06824 bad =       2
.
.
.
at    5   4 swapped in    19       4.72563 bad =       0
at    6   1 swapped in    11       4.77369 bad =       2
.
.
.
at    6   4 swapped in     7       4.88972 bad =       2
at    7   1 swapped in    16       4.88972 bad =       2
.
.
.
at    7   4 swapped in    24       5.05078 bad =       2
at    8   1 swapped in    10       5.05078 bad =       2
.
.
.
at    8   4 swapped in    30       5.08288 bad =       2
                        1          5.08288 *     0.19674
at    1   3 swapped in    23       5.11786 bad =       1
at    1   4 swapped in    25       5.12802 bad =       1
at    2   1 swapped in    21       5.12802 bad =       2
.
.
.
at    2   4 swapped in    26       5.37339 bad =       2
at    3   1 swapped in     3       5.07267 bad =       2
.
.
.
at    3   4 swapped in     6       5.07267 bad =       1
at    4   2 swapped in     1       5.07730 bad =       2
.
.
.
at    4   3 swapped in    18       5.31648 bad =       2
at    6   1 swapped in    11       5.31648 bad =       2
.
.
.
at    6   4 swapped in     9       5.44566 bad =       2
at    7   1 swapped in    16       5.44839 bad =       2
.
.
.
at    7   4 swapped in    24       5.47225 bad =       2
                        2          5.47225 *     0.18274
```

```
   .
   .
   .
                         8          5.51207       0.18142
   at    7   1 swapped in     6       5.51207 bad =       2
   at    7   2 swapped in     8       5.51207 bad =       2
   at    7   3 swapped in    17       5.51207 bad =       2
   at    7   4 swapped in    24       5.51207 bad =       2
                         9          5.51207       0.18142
   at    7   1 swapped in     6       5.51207 bad =       2
   at    7   2 swapped in     8       5.51207 bad =       2
   at    7   3 swapped in    17       5.51207 bad =       2
   at    7   4 swapped in    24       5.51207 bad =       2
                        10          5.51207       0.18142

WARNING: Design 1 has TYPES=, OPTIONS=NODUP, or restrictions problems.
```

The macro does not succeed in imposing the restrictions.

The design summary and the final efficiency results are as follows:

```
                          Final Results

                    Design                  1
                    Choice Sets             8
                    Alternatives            4
                    Parameters             18
                    Maximum Parameters     24
                    D-Efficiency       5.5121
                    Relative D-Eff    68.9009
                    D-Error            0.1814
                    1 / Choice Sets    0.1250
                    WARNING: Restriction violations.
```

Again, the output states that there are restriction violations. The following step displays the design:

```
proc print data=desres2; id set; by set; var x:; run;
```

The first two choice sets are as follows:

```
             Set    x1     x2     x3     x4     x5     x6

              1      4      1      2      1      1      2
                     2      1      1      4      3      3
                     3      1      4      3      1      1
                     1      1      3      2      3      4
```

```
                    2     2    4    2    2    2    4
                          3    3    1    2    3    1
                          1    2    4    3    3    2
                          1    1    2    4    4    1
```

Neither choice set has two constant attributes. The `%ChoicEff` macro is very much like the `%MktEx` macro in the way that you must quantify badness. It is often not sufficient to have a badness value that is zero when everything is fine and not zero when things are not fine. Consider `x1` in the second choice set. Imagine changing the 3 to a 2. That moves the attribute closer to constant but has no effect on the badness criterion. The badness criterion is only affected when an attribute with 3 values that are constant is changed to one with 4 values that are constant or an attribute with 4 values that are constant is changed to one with 3 values that are constant. The `%ChoicEff` macro needs to be provided with more guidance than this. Creating constant attributes decreases efficiency, so that is not a direction that the `%ChoicEff` macro tends to go unless you guide it in that direction.

The following steps illustrate one way to guide it:

```
%mktex(4 ** 6, n=128, seed=104, maxdesigns=1, options=nodups)

%macro res;
   c = 0;                         /* n of constant attrs in x          */
   do i = 1 to ncol(x);
      c = c +                     /* n of constant attrs in x          */
         all(x[,i] = round(x[:,i]));/* all values equal average value  */
      end;
   bad = bad + abs(c - 2);        /* want two attrs constant           */
   if c < 2 then do;              /* refine bad if we need more constants */
      do i = 1 to ncol(x);
         bad = bad +              /* count values not at the average   */
               sum(x[,i] ^= round(x[:,i]));
         end;
      end;
   %mend;

%choiceff(data=randomized,        /* candidate set of alternatives     */
          model=class(x1-x6 / sta), /* model with stdz orthogonal coding */
          nsets=8,                /* number of choice sets             */
          flags=4,                /* 4 alternatives, generic candidates */
          seed=104,               /* random number seed                */
          options=relative        /* display relative D-efficiency     */
                  resrep,         /* detailed report on restrictions   */
          restrictions=res,       /* name of the restrictions macro    */
          resvars=x1-x6,          /* variable names used in restrictions */
          maxiter=1,              /* maximum number of designs to make */
          bestout=desres3,        /* final choice design               */
          beta=zero)              /* assumed beta vector, Ho: b=0      */

proc print data=desres3; id set; by set; var x:; run;
```

In this example, the restrictions macro begins the same way as before, but when fewer than two attributes are constant, the badness value is increased by the number of values in each of the attributes that are not equal to the average. Now, when any value is changed to a 2 in `x1` in the second choice set, the badness criterion decreases. The `%ChoicEff` macro now knows when it is taking a step in the right direction even if it has not gotten to its ultimate goal yet. This is very important! If you do not let the `%ChoicEff` macro know when it is doing the right thing, it might not succeed in doing what you want. You want the `%ChoicEff` macro to move toward a restricted design with a lower efficiency. The `%ChoicEff` macro tries to move in another direction, towards a more efficient design. When you want the `%ChoicEff` macro to look somewhere than other where it would prefer to look, you need to tell it when it is moving in the right direction. One other change was made to make this example. The `%MktEx` macro was changed to make a larger candidate set, 128 candidates, none of which are duplicates of any other candidate.

Part of the output is as follows:

```
                          Final Results

                    Design                    1
                    Choice Sets               8
                    Alternatives              4
                    Parameters               18
                    Maximum Parameters       24
                    D-Efficiency         3.5824
                    Relative D-Eff      44.7803
                    D-Error              0.2791
                    1 / Choice Sets      0.1250

                Set    x1    x2    x3    x4    x5    x6

                  1     4     1     2     1     2     4
                        3     3     2     1     3     4
                        2     2     2     1     1     2
                        3     4     2     1     1     1

                  2     3     4     3     1     3     3
                        3     3     3     2     4     1
                        3     1     3     3     1     2
                        3     3     3     4     2     2

                  3     4     1     3     4     1     4
                        4     1     2     2     1     3
                        4     1     4     1     4     1
                        4     1     4     2     3     2
```

Now, the restrictions are all correctly imposed. This approach creates a design that is 45% efficient relative to the optimal design with no restrictions. Again, our concern at the moment is in evaluating our restrictions macro, not in finding the absolute maximum for *D*-efficiency.

The following steps jointly impose both sets of restrictions considered in this example so far and evaluate the design:

```
%mktex(4 ** 6, n=128, seed=104, maxdesigns=1, options=nodups)

%macro res;
   do i = 1 to nalts;
      do k = i + 1 to nalts;
         if all(x[i,] >= x[k,])     /* alt i dominates alt k               */
            then bad = bad + 1;
         if all(x[k,] >= x[i,])     /* alt k dominates alt i               */
            then bad = bad + 1;
         end;
      end;
   c = 0;                           /* n of constant attrs in x            */
   do i = 1 to ncol(x);
      c = c +                       /* n of constant attrs in x            */
         all(x[,i] = round(x[:,i]));/* all values equal average value      */
      end;
   bad = bad + abs(c - 2);          /* want two attrs constant             */
   if c < 2 then do;                /* refine bad if we need more constants */
      do i = 1 to ncol(x);
         bad = bad +                /* count values not at the average     */
               sum(x[,i] ^= round(x[:,i]));
         end;
      end;
   %mend;

%choiceff(data=randomized,          /* candidate set of alternatives       */
          model=class(x1-x6 / sta), /* model with stdz orthogonal coding   */
          nsets=8,                  /* number of choice sets               */
          flags=4,                  /* 4 alternatives, generic candidates  */
          seed=104,                 /* random number seed                  */
          options=relative          /* display relative D-efficiency       */
                  resrep,           /* detailed report on restrictions     */
          restrictions=res,         /* name of the restrictions macro      */
          resvars=x1-x6,            /* variable names used in restrictions  */
          maxiter=1,                /* maximum number of designs to make   */
          bestout=desres4,          /* final choice design                 */
          beta=zero)                /* assumed beta vector, Ho: b=0        */

proc print data=desres4; id set; by set; var x:; run;
proc iml;
   use desres4(keep=x1-x6); read all into x;
   sets = 8;
   alts = 4;
   if sets # alts ^= nrow(x) then print 'ERROR: Invalid sets and/or alts.';
   do a = 1 to sets;
      print a[label='Set'] '   '
            (x[((a - 1) * alts + 1):a * alts,])[format=1.] '          ';
      ii = 0;
      do i = (a - 1) * alts + 1 to a * alts;
         ii = ii + 1;
         kk = ii;
         do k = i + 1 to a * alts;
            kk = kk + 1;
            print ii[label='Alt'] '   ' (x[i,])[format=1.]
                  (sum(x[i,] >= x[k,]))[label='Sum'],
                  kk[label=none]  '   ' (x[k,])[format=1.]
                  (sum(x[k,] >= x[i,]))[label=none];
            if all(x[i,] >= x[k,]) | all(x[k,] >= x[i,]) then
               print "ERROR: Sum=0.";
            end;
         end;
      end;
   quit;
```

Part of the output is as follows:

```
                              Final Results

                        Design                    1
                        Choice Sets               8
                        Alternatives              4
                        Parameters               18
                        Maximum Parameters       24
                        D-Efficiency         3.5175
                        Relative D-Eff      43.9692
                        D-Error              0.2843
                        1 / Choice Sets      0.1250
```

This approach creates a design that is 44% efficient relative to the optimal design with no restrictions. Our concern at the moment is still in evaluating our restrictions macro, not in finding the absolute maximum for *D*-efficiency. We do not know what the maximum *D*-efficiency is for this design, but with 2 of 8 attributes constrained to be constant, the optimum value must be less than $100 \times 6/8 = 75\%$.

The first two choice sets are as follows:

```
            Set     x1     x2     x3     x4     x5     x6

              1      1      1      3      1      3      1
                     3      3      2      1      3      4
                     4      2      2      1      3      4
                     2      4      2      1      3      2

              2      3      1      2      1      1      4
                     3      1      4      3      3      2
                     3      1      4      2      4      2
                     3      1      1      3      2      3
```

The dominance evaluation results for the first choice set are as follows:

```
                    Set

                      1     1 1 3 1 3 1
                            3 3 2 1 3 4
                            4 2 2 1 3 4
                            2 4 2 1 3 2

                    Alt                       Sum

                      1     1 1 3 1 3 1         3
                      2     3 3 2 1 3 4         5

                    Alt                       Sum

                      1     1 1 3 1 3 1         3
                      3     4 2 2 1 3 4         5

                    Alt                       Sum

                      1     1 1 3 1 3 1         3
                      4     2 4 2 1 3 2         5

                    Alt                       Sum

                      2     3 3 2 1 3 4         5
                      3     4 2 2 1 3 4         5

                    Alt                       Sum

                      2     3 3 2 1 3 4         5
                      4     2 4 2 1 3 2         4
```

```
                             Alt                         Sum

                               3     4 2 2 1 3 4           5
                               4     2 4 2 1 3 2           4
```

The design conforms to all restrictions in every choice set.

Now that we are certain that the design conforms to the restrictions, we can try to find a more efficient design. The following step creates a full-factorial candidate set with $4^6 = 4096$ candidate alternatives and specifies `maxiter=10` in the `%ChoiceEff` macro:

```
%mktex(4 ** 6, n=4096, seed=104)

%macro res;
   do i = 1 to nalts;
      do k = i + 1 to nalts;
         if all(x[i,] >= x[k,])     /* alt i dominates alt k              */
            then bad = bad + 1;
         if all(x[k,] >= x[i,])     /* alt k dominates alt i              */
            then bad = bad + 1;
         end;
      end;
   c = 0;                           /* n of constant attrs in x           */
   do i = 1 to ncol(x);
      c = c +                       /* n of constant attrs in x           */
         all(x[,i] = round(x[:,i]));/* all values equal average value     */
      end;
   bad = bad + abs(c - 2);          /* want two attrs constant            */
   if c < 2 then do;                /* refine bad if we need more constants */
      do i = 1 to ncol(x);
         bad = bad +                /* count values not at the average    */
               sum(x[,i] ^= round(x[:,i]));
         end;
      end;
   %mend;

%choiceff(data=randomized,          /* candidate set of alternatives      */
          model=class(x1-x6 / sta), /* model with stdz orthogonal coding  */
          nsets=8,                  /* number of choice sets              */
          flags=4,                  /* 4 alternatives, generic candidates */
          seed=104,                 /* random number seed                 */
          rscale=8 * 6 / 8,         /* relative D-efficiency scale factor */
                                    /* 6 of 8 attrs in 8 sets vary        */
          options=resrep,           /* detailed report on restrictions    */
          restrictions=res,         /* name of the restrictions macro     */
          resvars=x1-x6,            /* variable names used in restrictions */
          maxiter=10,               /* maximum number of designs to make  */
          bestout=desres5,          /* final choice design                */
          beta=zero)                /* assumed beta vector, Ho: b=0       */
```

```
proc print data=desres5; id set; by set; var x:; run;
```

The `rscale=` option specifies that relative *D*-efficiency is not scaled relative to 8 choice sets. Rather, it is scaled relative to a value that is three-quarters as large since only six of eight attributes can vary in each choice set.

Part of the output is as follows:

```
                    Final Results

             Design                     1
             Choice Sets                8
             Alternatives               4
             Parameters                18
             Maximum Parameters        24
             D-Efficiency          4.6354
             Relative D-Eff       77.2566
             D-Error               0.2157
             1 / Choice Sets       0.1250
```

This approach takes much longer than the previous approach. The design is better than the one found previously (unscaled *D*-efficiency of 4.64 compared to 3.52 previously). This approach creates a design that is 77% efficient relative to the optimal design with 6 of 8 attributes varying. It is 100 × 4.6354/8 = 57.94% *D*-efficient relative to the optimal design with no restrictions (compared to 43.7% found previously with the smaller candidate set and fewer iterations).

The final design is as follows:

```
Set    x1    x2    x3    x4    x5    x6

 1      1     3     4     1     2     4
        3     1     2     4     2     4
        4     4     3     2     2     4
        2     2     1     3     2     4

 2      2     3     2     2     1     3
        2     1     3     2     3     1
        2     2     2     2     4     4
        2     4     1     2     2     2

 3      3     4     3     2     3     3
        1     2     3     4     2     3
        2     1     3     4     1     3
        4     3     3     3     4     3

 4      4     4     2     3     3     2
        4     4     1     2     1     4
        4     4     3     1     2     3
        4     4     4     4     4     1
```

```
5       2     2     2     2     2     2
        4     2     1     1     3     2
        1     2     4     3     1     2
        3     2     3     1     4     2

6       4     2     2     4     1     2
        1     1     2     4     4     3
        3     3     2     4     2     1
        2     4     2     4     3     4

7       2     3     3     3     1     2
        4     1     4     3     1     4
        3     2     4     3     1     3
        1     4     1     3     1     1

8       2     1     4     2     2     2
        2     2     3     3     2     4
        2     4     2     1     2     1
        2     3     1     4     2     3
```

The results of the PROC IML step that evaluates the design (not shown) show that the design conforms to all of the restrictions.

Restrictions Within and Across Choice Sets

This example uses a fairly complicated restrictions macro. The goal is to avoid dominated alternatives like before. Another goal is to require certain patterns of constant attributes. Each choice set is required to have one or two constant attributes. Furthermore, each attribute is required to be constant within a specified number of choice sets. Specifically, attributes 1, 3, and 5 are required to be constant in two choice sets and attributes 2, 4, and 6 are required to be constant in one choice set. This last requirement requires more than just simple variations on the technique shown previously.

In this example, like the last example, restrictions are formulated based on `x`, the candidate choice set. However, this example also has restrictions that are defined across choice sets not just within a choice set. Therefore, restrictions are also defined based on `xmat`, the full design. The macro provides you with the value of an index variable, `setnum`, that contains the number of the choice set being worked on. The choice set `x` corresponds to the value of `setnum`. For example, when `setnum` = 3, then `x` is the third choice set. The `setnum` choice set in `xmat` is currently in the design, and the choice set in `x` is being considered as a replacement for it. The scalar `altnum` is available as well, and it contains the number of the alternative that is being changed. This scalar is only available when you are using the algorithm that swaps candidate alternatives. It is not available when you provide a candidate set of choice sets. Two additional scalars are available for you to use: `nalts`, the number of alternatives in the design and `nsets`, the number of choice sets in the design. Using these scalars rather than hard-coded constants makes it easier for you to modify a macro for use in a different situation.

The following step creates a candidate set of 256 alternatives with no duplicates:

```
%mktex(4 ** 6, n=256, seed=104, maxdesigns=1, options=nodups)
```

The following step creates the restrictions macro:

```
%macro res;
   do i = 1 to nalts;
      do k = i + 1 to nalts;
         if all(x[i,] >= x[k,])     /* alt i dominates alt k               */
            then bad = bad + 1;
         if all(x[k,] >= x[i,])     /* alt k dominates alt i               */
            then bad = bad + 1;
         end;
      end;

   nattrs = ncol(x);                /* number of columns in design         */
   v = j(1, nattrs, 0);             /* n of constant attrs across sets     */
   c = 0;                           /* n of constant attrs within set      */

   do i = 1 to nattrs;              /* loop over all attrs                 */
      a    = all(x[,i] = x[1,i]);   /* 1 - attr i constant, 0 - varying    */
      c    = c + a;                 /* n of constant attrs within set      */
      v[i] = v[i] + a;              /* n of constant attrs across sets     */
      end;

   if c > 2 | c = 0 then            /* want 1 or 2 constant attrs in a set  */
      bad = bad + 10 # abs(c - 2);  /* weight of 10 prevents trade offs    */

   do s = 1 to nsets;               /* loop over rest of design            */
      if s ^= setnum then do;       /* skip xmat part that corresponds to x */
         z = xmat[((s-1)*nalts+1) : /* pull out choice set s               */
                  (s * nalts),];
         do i = 1 to nattrs;        /* loop over attrs                     */
            v[i] = v[i] +           /* n of constant attrs across sets     */
                   all(z[,i] = z[1,i]);
            end;
         end;
      end;

   d   = abs(v - {2 1 2 1 2 1})[+]; /* see if constant attrs match target  */
   bad = bad + d;                   /* increase badness                    */
   if d then do;                    /* if not at target, fine tune badness */
      do i = 1 to nattrs;           /* loop over attrs                     */
         bad = bad +                /* add to badness as attrs are farther */
              (x[,i] ^= x[1,i])[+];/* from constant                       */
         end;
      end;
   %mend;
```

Note that `v` is a vector with m elements, one for each attribute. The statements `v[i] = v[i] + a` and `v[i] = v[i] + all(z[,i] = z[1,i])` add one to the *ith* element of `v` whenever the *ith* attribute in `x` or `xmat` is constant. Now consider the statement: `d = abs(v - {2 1 2 1 2 1})[+]`. The expression `abs(v - {2 1 2 1 2 1})` creates a vector with m elements containing the absolute differences between the counts of the number of constant attributes and the target counts. The subscript reduction operator `[+]` adds up the absolute differences, then the result is stored in `d`. When `d` is zero, the right number of attributes are constant across choice sets. Otherwise, `d` is a measure of how far the design is from conforming to the restrictions. The measure of design badness must be more sensitive than this. When `d` is not zero, badness is increased for every value that is different from constant. This way, the `%ChoicEff` macro knows when it is moving in the right direction toward making more constant attributes. The statement `bad = bad + (x[,i] ^= x[1,i])[+]` creates a vector `(x[,i] ^= x[1,i])` with ones for all values that are not equal to the first value and zeros for values that are equal. Then the values in this vector are summed by the subscript reduction operator to provide a count of the number of values not equal to the first value, and badness is increased by that amount.

The statement `bad = bad + 10 # abs(c - 2)` weights the number of constant attributes within a choice set with a weight of 10.* This weight is greater than the implicit weights of one for the other components of the badness function. This means that minimizing this source of badness takes precedence over minimizing other sources, and there won't be any trade offs between this source and other sources. Sometimes, when there are multiple sources of badness, it is important to differentially weight them. It might not be important how you weight them or which source gets the most weight. Simply providing some differential weighting helps the `%ChoicEff` macro figure out how to impose all of the restrictions. Otherwise the `%ChoicEff` macro might get stuck increasing one source of badness every time it decreases another. Weighting can also help you interpret `options=resrep` results.

The following step searches the candidate set of alternatives and creates the restricted choice design:

```
%choiceff(data=randomized,         /* candidate set of alternatives       */
          model=class(x1-x6 / sta), /* model with stdz orthogonal coding   */
          nsets=8,                 /* number of choice sets               */
          flags=4,                 /* 4 alternatives, generic candidates  */
          seed=104,                /* random number seed                  */
          options=relative         /* display relative D-efficiency       */
                  resrep,          /* detailed report on restrictions     */
          restrictions=res,        /* name of the restrictions macro      */
          resvars=x1-x6,           /* variable names used in restrictions */
          maxiter=1,               /* maximum number of designs to make   */
          bestout=desres6,         /* final choice design                 */
          beta=zero)               /* assumed beta vector, Ho: b=0        */
```

*The IML operator # performs scalar multiplication.

Part of the output is as follows:

```
                        Final Results

                  Design                    1
                  Choice Sets               8
                  Alternatives              4
                  Parameters               18
                  Maximum Parameters       24
                  D-Efficiency         5.0372
                  Relative D-Eff      62.9645
                  D-Error              0.1985
                  1 / Choice Sets      0.1250
```

This step creates a design that is 63% efficient relative to an optimal design with no restrictions.

The following step displays the design:

```
   proc print data=desres6; id set; by set; var x:; run;
```

The results are as follows:

```
             Set    x1    x2    x3    x4    x5    x6

              1      4     3     1     4     1     3
                     1     3     1     2     4     1
                     3     3     1     4     3     4
                     4     3     1     1     2     2

              2      4     3     1     3     3     4
                     2     1     2     3     3     3
                     1     4     3     3     4     2
                     3     1     4     3     1     3

              3      4     2     4     3     3     3
                     4     3     2     2     2     1
                     4     4     1     1     2     4
                     4     1     3     4     4     2

              4      2     3     4     4     1     4
                     3     1     2     3     4     4
                     3     2     3     1     3     4
                     1     2     4     4     2     4

              5      4     2     4     2     4     3
                     3     4     4     3     1     1
                     2     3     4     4     2     3
                     1     4     4     2     3     4
```

```
 6        2     2     3     3     4     4
          1     3     2     1     4     3
          3     1     1     4     4     1
          4     4     4     2     4     2

 7        2     3     4     1     4     4
          2     1     1     3     2     3
          2     4     2     4     4     1
          2     2     1     2     1     2

 8        3     4     1     2     4     3
          1     3     2     3     4     2
          4     1     3     4     4     1
          2     2     4     1     4     4
```

Choice set one has 2 constant attributes and choice sets 2 through 8 have 1 constant attribute. Choice sets 1, 3, and 5 have two constant attributes within choice set. Choice sets 2, 4, and 6 have one constant attributes within choice set. No alternatives are dominated. Now that we are certain that our restrictions macro is correct, we can make a full-factorial candidate set as follows:

```
%mktex(4 ** 6, n=4096)
```

Using the same restrictions macro and `%ChoicEff` macro call as before (one iteration), the results are as follows:

```
                    Final Results

                Design                  1
                Choice Sets             8
                Alternatives            4
                Parameters             18
                Maximum Parameters     24
                D-Efficiency       5.6758
                Relative D-Eff    70.9478
                D-Error            0.1762
                1 / Choice Sets    0.1250
```

The *D*-efficiency is 71% compared to 63% previously. With more iteration (increasing the value of `maxiter=`), you would expect that value to increase.

There are many things that can go wrong when you are creating restricted choice designs. You might write a restrictions macro that is internally contradictory or otherwise write a set of restrictions that cannot possibly be satisfied. The `%ChoicEff` macro cannot analyze these problems for you, but it can tell you when it fails to meet restrictions. You might write a restrictions macro that is correct but fails to provide the `%ChoicEff` macro the guidance that it needs. You might instead create a candidate set that is too small and limited. You might need to create a larger candidate set so that the macro has more freedom to find an efficient design. However, you do not want to start with a candidate set that is too large at first, because it takes a long time to search large candidate sets, particularly when there are restrictions. Often there is some trial and error involved in creating the right restrictions macro and the right set of candidates.

Restrictions Within and Across Choice Sets with Candidate Set Swapping

This next example tackles the same problem as the previous example, but this time we use the approach of creating a candidate set of choice sets instead of a candidate set of alternatives. This next example is *not* the recommended approach for this example or most other examples now that the `restrictions=` option is implemented in the `%ChoicEff` macro. It is simply provided here for completeness and because it illustrates important differences between the two approaches. Most of the previous examples created a candidate set of alternatives. With 6 four-level factors, there are $4^6 = 4096$ possible candidate alternatives. From those 4096 possible alternatives, all of the $4096^4 = 281,474,976,710,656$ (281 trillion) possible choice sets can potentially be constructed. In practice, only a tiny fraction of them are considered (perhaps several thousand or a few million), but all are possible. In contrast, in the choice set swapping algorithm, you must create a candidate set of choice sets, so only a few thousand choice sets at the most can be considered for most problems. It is all but guaranteed that the alternative swapping algorithm will do better than the choice set swapping algorithm.

You begin using the candidate set swapping algorithm by creating a candidate set of choice sets. You need to impose the within-choice-set restrictions at this point. The following steps create a candidate set of choice sets:

```
%macro res2;

   nattrs = 6;                      /* 6 attributes                        */
   nalts  = 4;                      /* 4 alternatives                      */
   z = shape(x, nalts, nattrs);     /* rearrange x to look like a choice set*/

   do ii = 1 to nalts;
      do k = ii + 1 to nalts;
         if all(z[ii,] >= z[k,])    /* alt ii dominates alt k              */
            then bad = bad + 1;
         if all(z[k,] >= z[ii,])    /* alt k dominates alt ii              */
            then bad = bad + 1;
         end;
      end;

   c = 0;                           /* n of constant attrs within set      */
   do ii = 1 to nattrs;             /* loop over all attrs                 */
      c = c +                       /* n of constant attrs within set      */
          all(z[,ii] = z[1,ii]);    /* 1 - attr i constant, 0 - varying    */
      end;
   if c > 2 | c = 0 then            /* want 1 or 2 constant attrs in a set  */
      bad = bad + 10 # abs(c - 2);  /* weight of 10 prevents trade offs    */
   %mend;
```

```
%mktex(4 ** 24, n=200, restrictions=res2, seed=104,
       target=90, options=quickr resrep, order=random)

%mktkey(4 6)

%mktroll(design=randomized, key=key, out=rolled)
```

The restrictions macro is very similar to the within-choice-set part of the previous restrictions macro. The first difference is due to the fact that the `%MktEx` macro is creating a linear arrangement of the attributes—one in which all attributes of all alternatives are arranged in a single row. The statement `z = shape(x, nalts, nattrs)` rearranges `x` into a matrix with one row for each alternative and one column for each attribute. This is not necessary, but it enables us to impose the restrictions using similar code to that which was used previously (only now based on `z` instead of `x`). Another difference is that the index `i` had been replaced with `ii`. In a `%MktEx` macro restrictions macro, `i` is the row number being worked on and you cannot change it.

The `%MktEx` macro makes a design with 24 factors (four alternatives times six attributes). It specifies the name of the restrictions macro in the `restrictions=res2` option. The option `target=90` specifies that iteration can stop when all restrictions are met and the design is 90% efficient. Since this is a candidate set, we do not need to maximize *D*-efficiency at this stage. The option `options=quickr` makes one design quickly using the coordinate exchange algorithm and a random initialization. The option `options=resrep` provides a detailed report of how the design is conforming to the restrictions. The option `order=random` loops over the columns in a random order. A candidate set of choice sets with 80 candidates is created. The `%MktKey` macro creates the rules for turning a linear arrangement into a choice design, and the `%MktKey` macro creates the candidate set of choice sets.

The following step creates the restrictions macro for the `%ChoicEff` macro:

```
%macro res;
   nattrs = ncol(x);              /* number of columns in design         */
   v = j(1, nattrs, 0);           /* n of constant attrs across sets     */
   do s = 1 to nsets;             /* loop over each choice set           */
      if s ^= setnum then         /* pull choice set out of xmat         */
         z = xmat[((s - 1) * nalts + 1) : (s * nalts),];
      else z = x;                 /* if current set, get from x          */
      do i = 1 to nattrs;         /* loop over attrs                     */
         v[i] = v[i] +            /* n of constant attrs across sets     */
                all(z[,i] = z[1,i]);
         end;
      end;
   bad = abs(v - {2 1 2 1 2 1})[+]; /* see if constant attrs match target   */
   %mend;
```

Since the candidate set of choice sets already has the within-choice-set restrictions imposed, only the between-choice-set restrictions are imposed. Furthermore, the code at the end of the macro (`if d then do; do i = 1 to nattrs; bad = bad + (x[,i] ^= x[1,i])[+]; end; end;`) is removed. There is no opportunity to refine a candidate choice set. It either conforms to the restrictions or it does not. That code only serves to obfuscate the badness criterion in this example with the candidate choice set search algorithm.

The following step searches for the design:

```
%choiceff(data=rolled,             /* candidate set of choice sets         */
          model=class(x1-x6 / sta), /* model with stdz orthogonal coding    */
          nsets=8,                  /* number of choice sets                */
          nalts=4,                  /* number of alternatives               */
          seed=104,                 /* random number seed                   */
          options=relative          /* display relative D-efficiency        */
                  resrep,           /* detailed report on restrictions      */
          restrictions=res,         /* name of the restrictions macro       */
          resvars=x1-x6,            /* variable names used in restrictions  */
          maxiter=2,                /* maximum number of designs to make    */
          bestout=desres7,          /* final choice design                  */
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

Since a candidate set of choice sets is searched and not a candidate set of alternatives, the **nalts=** option is used instead of the **flags=** option. Some of the results are as follows:

```
                              Final Results

                        Design                    2
                        Choice Sets               8
                        Alternatives              4
                        Parameters               18
                        Maximum Parameters       24
                        D-Efficiency         4.3467
                        Relative D-Eff      54.3337
                        D-Error              0.2301
                        1 / Choice Sets      0.1250
```

The design is 54% efficient compared to 71% previously. With 400 candidate choice sets (not shown) the design is 56% efficient.

The following steps display and evaluate the design:

```
proc print data=desres7; id set; by notsorted set; var x:; run;
proc iml;
   use desres7(keep=x1-x6); read all into x;
   sets = 8;
   alts = 4;
   if sets # alts ^= nrow(x) then print 'ERROR: Invalid sets and/or alts.';
   do a = 1 to sets;
      print a[label='Set'] '   '
            (x[((a - 1) * alts + 1):a * alts,])[format=1.] '          ';
      ii = 0;
      do i = (a - 1) * alts + 1 to a * alts;
         ii = ii + 1;
         kk = ii;
         do k = i + 1 to a * alts;
            kk = kk + 1;
            print ii[label='Alt'] '   ' (x[i,])[format=1.]
                  (sum(x[i,] >= x[k,]))[label='Sum'],
                  kk[label=none]  '   ' (x[k,])[format=1.]
                  (sum(x[k,] >= x[i,]))[label=none];
            if all(x[i,] >= x[k,]) | all(x[k,] >= x[i,]) then
               print "ERROR: Sum=0.";
            end;
         end;
      end;
   quit;
```

The design is as follows:

```
                     Set    x1     x2     x3     x4     x5     x6

                      56     3      4      4      2      3      1
                             2      2      3      2      1      2
                             1      2      1      2      2      3
                             3      4      2      2      4      2

                     106     2      4      2      1      2      4
                             4      2      2      2      1      4
                             2      3      1      3      3      4
                             1      3      3      4      4      4

                      92     2      3      4      1      2      3
                             2      2      4      2      3      1
                             2      4      3      3      4      4
                             2      1      1      4      3      2

                      96     1      4      2      4      1      3
                             4      1      2      2      2      4
                             2      2      2      1      1      1
                             1      3      2      1      3      2
```

```
     185       1       4       3       4       4       4
               4       3       1       1       4       2
               2       3       2       2       4       3
               4       1       2       3       4       1

     131       4       4       2       4       4       1
               1       4       4       1       4       2
               3       4       4       2       4       1
               3       4       2       3       4       3

      34       4       2       3       1       1       1
               3       2       3       4       2       4
               2       1       3       1       3       3
               1       4       3       2       4       2

     182       1       2       4       3       4       3
               1       3       3       4       2       1
               1       2       2       1       4       4
               1       3       4       3       1       1
```

The design conforms to all restrictions, but again, this is not the recommended approach for this problem.

Brand Effects

This next example creates a design with a brand factor. There are three brands, three additional attributes, and three alternatives. A choice set has nine values: three alternatives times three attributes. The goal is to restrict the design so that each level occurs between three and five times in each of the nine positions across all choice sets. In other words, the goal is to ensure a nearly balanced choice design.

The following steps make and display a candidate set of alternatives:

```
%mktex(3 ** 3, n=3**3)

data full;
   Set + 1;
   Brand = 0;
   set design;
   retain f1-f3 0;
   array f[3];
   do brand = 1 to 3;
      f[brand] = 1; output; f[brand] = 0;
      end;
   run;

proc print; id set; by set; run;
```

A few of the candidate alternatives are as follows:

```
Set    Brand    x1    x2    x3    f1    f2    f3

  1      1       1     1     1     1     0     0
         2       1     1     1     0     1     0
         3       1     1     1     0     0     1

  2      1       1     1     2     1     0     0
         2       1     1     2     0     1     0
         3       1     1     2     0     0     1

  3      1       1     1     3     1     0     0
         2       1     1     3     0     1     0
         3       1     1     3     0     0     1
  .
  .
  .
 27      1       3     3     3     1     0     0
         2       3     3     3     0     1     0
         3       3     3     3     0     0     1
```

The `%MktEx` macro makes the full-factorial candidate set of alternatives for all of the attributes except brand. The DATA step adds a choice set number, which is not needed, but it is useful for displaying the candidate set. Three brands are also added. The initial `Brand = 0` statement positions the brand variable after the set variable and before the other variables and sets the case that SAS uses to display the name. Three flag variables are added to the design. Brand one alternatives are flagged by `f1 = 1 f2 = 0 f3 = 0`, brand two alternatives are flagged by `f1 = 0 f2 = 1 f3 = 0`, and brand three alternatives are flagged by `f1 = 0 f2 = 0 f3 = 1`. The flag variables are retained so that the all zero values from the previous candidate are available as initial values for the next candidate. Each candidate alternative is written out three times, once for each brand.

The following step creates the restrictions macro:

```
%macro res;
   c = j(9, 3, 0);                  /* counts - nine positions x 3 levels   */
   do s = 1 to nsets;               /* loop over sets                       */
      if s = setnum then z = x;     /* get choice set from x or xmat        */
      else z = xmat[((s - 1) # nalts + 1) : s # nalts,];
      k = 0;                        /* index into count matrix c            */
      do j = 1 to ncol(z);          /* loop over attributes                 */
         do i = 1 to nalts;         /* loop over alternatives               */
            k = k + 1;              /* index into the next row of c         */
            a = z[i,j];             /* index into c for the z[i,j] level    */
            c[k,a] = c[k,a] + 1;    /* add one to count                     */
            end;
         end;
      end;
   bad = sum((c < 3) # abs(c - 3)) +/* penalty for counts being less than 3 */
         sum((c > 5) # abs(c - 5)); /* penalty for counts greater than 5    */
   %mend;
```

A matrix `c` of counts is initialized to zero. It has nine rows for each of the nine positions in a choice set and three columns for the three levels of each attribute. The `do` statement loops over all of the choice sets. When the index `s` matches the choice set being worked on, the current choice set `x` is stored in `z`. Otherwise, the relevant choice set is extracted from `xmat` and is stored in `z`. The next two `do` statements loop over the nine positions in the choice set. The positions are indexed by `k`. When the level is `a` ($a = 1, 2, 3$) then the *ath* column of `c` is incremented by one to count how often the *ath* level occurs in each of the nine positions. Badness is a function of the number of counts less than three and the number of counts greater than five. Specifically, the operation `(c < 3)` produces a matrix of zeros and ones. When the (i, j) element of c is less than three, the (i, j) element of `(c < 3)` is 1, and it is zero otherwise. When the (i, j) element of `c` is less than three, the (i, j) element of `(c < 3) # abs(c - 3)` is the absolute deviation between `c[i,j]` and three, and it is zero otherwise. The operation `#` performs element-wise multiplication. The sum of the elements in `(c < 3) # abs(c - 3)` and `(c > 5) # abs(c - 5)` provides a measure of how far the design is from having values in the right range.

The following step searches for the design:

```
%choiceff(data=full,                /* candidate set of alternatives        */
          model=class(brand)        /* brand effects                        */
                class(brand*x1      /* alternative-specific effects         */
                      brand*x2
                      brand*x3 /
                      zero=' '),    /* use all brands in these effects      */
          nsets=12,                 /* number of choice sets                */
          seed=104,                 /* random number seed                   */
          options=resrep,           /* detailed report on restrictions      */
          restrictions=res,         /* name of the restrictions macro       */
          resvars=x1-x3,            /* variable names used in restrictions  */
          flags=f1-f3,              /* flag which alt can go where, 3 alts  */
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

Some of the results are as follows:

```
                              Final Results

                          Design                 2
                          Choice Sets           12
                          Alternatives           3
                          Parameters            20
                          Maximum Parameters    24
                          D-Efficiency      0.5163
                          D-Error           1.9369


          Variable                                                 Standard
     n    Name          Label               Variance     DF          Error

     1    Brand1        Brand 1              7.57495      1        2.75226
     2    Brand2        Brand 2              7.16595      1        2.67693
     3    Brand1x11     Brand 1 * x1 1       3.24298      1        1.80083
     4    Brand1x12     Brand 1 * x1 2       2.85037      1        1.68830
     5    Brand2x11     Brand 2 * x1 1       3.05369      1        1.74748
     6    Brand2x12     Brand 2 * x1 2       2.73221      1        1.65294
     7    Brand3x11     Brand 3 * x1 1       3.13116      1        1.76951
     8    Brand3x12     Brand 3 * x1 2       3.38122      1        1.83881
     9    Brand1x21     Brand 1 * x2 1       3.07311      1        1.75303
    10    Brand1x22     Brand 1 * x2 2       2.74792      1        1.65769
    11    Brand2x21     Brand 2 * x2 1       3.19973      1        1.78878
    12    Brand2x22     Brand 2 * x2 2       2.85195      1        1.68877
    13    Brand3x21     Brand 3 * x2 1       2.79820      1        1.67278
    14    Brand3x22     Brand 3 * x2 2       3.32705      1        1.82402
    15    Brand1x31     Brand 1 * x3 1       3.31434      1        1.82053
    16    Brand1x32     Brand 1 * x3 2       2.75006      1        1.65833
    17    Brand2x31     Brand 2 * x3 1       2.90460      1        1.70429
    18    Brand2x32     Brand 2 * x3 2       3.23177      1        1.79771
    19    Brand3x31     Brand 3 * x3 1       3.33634      1        1.82656
    20    Brand3x32     Brand 3 * x3 2       2.46393      1        1.56969
                                                         ==
                                                         20
```

The following step displays the design:

```
proc print; var brand x:; id set; by set; run;
```

The results are as follows:

```
                               Set    Brand     x1     x2     x3

                                 1      1        1      3      2
                                        2        2      3      1
                                        3        3      3      3

                                 2      1        3      3      2
                                        2        3      2      3
                                        3        1      3      2

                                 3      1        3      2      1
                                        2        2      3      3
                                        3        1      1      3

                                 4      1        2      2      3
                                        2        1      3      1
                                        3        1      3      1

                                 5      1        3      3      3
                                        2        2      2      2
                                        3        2      1      1

                                 6      1        2      3      1
                                        2        3      3      2
                                        3        2      2      2

                                 7      1        2      1      2
                                        2        3      1      1
                                        3        2      1      3

                                 8      1        3      1      3
                                        2        3      1      3
                                        3        3      2      1

                                 9      1        2      3      3
                                        2        2      1      1
                                        3        1      1      2

                                10      1        1      2      3
                                        2        3      2      1
                                        3        1      2      3

                                11      1        1      2      2
                                        2        1      1      2
                                        3        3      1      2

                                12      1        1      1      1
                                        2        1      2      3
                                        3        2      3      2
```

The following step evaluates the design:

```
proc iml;
   nsets = 12;  nalts = 3;
   use best(keep=x:); read all into xmat;
   c = j(9, 3, 0);                 /* counts - nine positions x 3 levels   */
   do s = 1 to nsets;              /* loop over sets                       */
      z = xmat[((s - 1) # nalts + 1) : s # nalts,];
      k = 0;                       /* index into count matrix c            */
      do j = 1 to ncol(z);         /* loop over attributes                 */
         do i = 1 to nalts;        /* loop over alternatives               */
            k = k + 1;             /* index into next row of c             */
            a = z[i,j];            /* index into c for the z[i,j] level    */
            c[k,a] = c[k,a] + 1;   /* add one to count                     */
            end;
         end;
      end;
   print c[format=2.];
   quit;
```

The results are as follows:

```
                                        c

                                     4  4  4
                                     3  4  5
                                     5  4  3
                                     3  4  5
                                     4  4  4
                                     5  3  4
                                     3  4  5
                                     5  3  4
                                     3  5  4
```

All of the counts are in the right range. In each position, the counts always sum to 12, the number of choice sets.

You can force all of the counts to be exactly four as follows:

```
%macro res;
   c = j(9, 3, 0);                 /* counts - nine positions x 3 levels   */
   do s = 1 to nsets;              /* loop over sets                       */
      if s = setnum then z = x;    /* get choice set from x or xmat        */
      else z = xmat[((s - 1) # nalts + 1) : s # nalts,];
      k = 0;                       /* index into count matrix c            */
      do j = 1 to ncol(z);         /* loop over attributes                 */
         do i = 1 to nalts;        /* loop over alternatives               */
            k = k + 1;             /* index into the next row of c         */
            a = z[i,j];            /* index into c for the z[i,j] level    */
            c[k,a] = c[k,a] + 1;   /* add one to count                     */
            end;
         end;
      end;
   bad = sum(abs(c - 4));          /* penalty for counts not at 4          */
   %mend;

%choiceff(data=full,               /* candidate set of alternatives        */
          model=class(brand)       /* brand effects                        */
                class(brand*x1     /* alternative-specific effects         */
                      brand*x2
                      brand*x3 /
                      zero=' '),   /* use all brands in these effects      */
          nsets=12,                /* number of choice sets                */
          seed=104,                /* random number seed                   */
          options=resrep,          /* detailed report on restrictions      */
          restrictions=res,        /* name of the restrictions macro       */
          resvars=x1-x3,           /* variable names used in restrictions  */
          flags=f1-f3,             /* flag which alt can go where, 3 alts  */
          beta=zero)               /* assumed beta vector, Ho: b=0         */

proc iml;
   nsets = 12;  nalts = 3;
   use best(keep=x:); read all into xmat;
   c = j(9, 3, 0);                 /* counts - nine positions x 3 levels   */
   do s = 1 to nsets;              /* loop over sets                       */
      z = xmat[((s - 1) # nalts + 1) : s # nalts,];
      k = 0;                       /* index into count matrix c            */
      do j = 1 to ncol(z);         /* loop over attributes                 */
         do i = 1 to nalts;        /* loop over alternatives               */
            k = k + 1;             /* index into next row of c             */
            a = z[i,j];            /* index into c for the z[i,j] level    */
            c[k,a] = c[k,a] + 1;   /* add one to count                     */
            end;
         end;
      end;
   print c[format=2.];
   quit;
```

Some of the results are as follows:

```
                              Final Results

                        Design                  2
                        Choice Sets            12
                        Alternatives            3
                        Parameters             20
                        Maximum Parameters     24
                        D-Efficiency       0.3688
                        D-Error            2.7117

           Variable                                                Standard
      n    Name         Label                Variance    DF           Error

      1    Brand1       Brand 1                7.9094     1         2.81236
      2    Brand2       Brand 2               37.7097     1         6.14083
      3    Brand1x11    Brand 1 * x1 1        22.2501     1         4.71700
      4    Brand1x12    Brand 1 * x1 2         9.8908     1         3.14497
      5    Brand2x11    Brand 2 * x1 1         4.1258     1         2.03121
      6    Brand2x12    Brand 2 * x1 2        13.0844     1         3.61724
      7    Brand3x11    Brand 3 * x1 1         5.6523     1         2.37746
      8    Brand3x12    Brand 3 * x1 2         9.0392     1         3.00653
      9    Brand1x21    Brand 1 * x2 1         7.2808     1         2.69830
     10    Brand1x22    Brand 1 * x2 2        11.7998     1         3.43508
     11    Brand2x21    Brand 2 * x2 1        13.1864     1         3.63130
     12    Brand2x22    Brand 2 * x2 2         8.2109     1         2.86546
     13    Brand3x21    Brand 3 * x2 1         3.0455     1         1.74514
     14    Brand3x22    Brand 3 * x2 2        11.8138     1         3.43713
     15    Brand1x31    Brand 1 * x3 1         7.0919     1         2.66305
     16    Brand1x32    Brand 1 * x3 2         5.9528     1         2.43983
     17    Brand2x31    Brand 2 * x3 1        13.6297     1         3.69185
     18    Brand2x32    Brand 2 * x3 2         9.2488     1         3.04119
     19    Brand3x31    Brand 3 * x3 1         5.0102     1         2.23835
     20    Brand3x32    Brand 3 * x3 2        11.5376     1         3.39670
                                                         ==
                                                         20
                              c

                           4  4  4
                           4  4  4
                           4  4  4
                           4  4  4
                           4  4  4
                           4  4  4
                           4  4  4
                           4  4  4
                           4  4  4
```

It is instructive to compare the variances of the parameter estimates for the two designs. In the second design, when the levels are more constrained, the variances are much larger and more variable. This is usually the sign of a bad design. There is a risk with constraints that you are hurting the efficiency and hence inflating some or all of the variances. For comparison, you can find the efficiency for the unconstrained design as follows:

```
   %choiceff(data=full,                /* candidate set of alternatives        */
             model=class(brand)        /* brand effects                        */
                   class(brand*x1      /* alternative-specific effects         */
                         brand*x2
                         brand*x3 /
                         zero=' '),    /* use all brands in these effects      */
             nsets=12,                 /* number of choice sets                */
             seed=104,                 /* random number seed                   */
             flags=f1-f3,              /* flag which alt can go where, 3 alts  */
             beta=zero)                /* assumed beta vector, Ho: b=0         */
```

Some of the results are as follows:

```
                              Final Results

                        Design                  1
                        Choice Sets            12
                        Alternatives            3
                        Parameters             20
                        Maximum Parameters     24
                        D-Efficiency       0.5099
                        D-Error            1.9611

           Variable                                                Standard
      n    Name          Label              Variance    DF         Error

      1    Brand1        Brand 1             8.51332     1       2.91776
      2    Brand2        Brand 2             8.90879     1       2.98476
      3    Brand1x11     Brand 1 * x1 1      3.34578     1       1.82915
      4    Brand1x12     Brand 1 * x1 2      3.65006     1       1.91051
```

```
             5    Brand2x11    Brand 2 * x1 1     3.35948     1      1.83289
             6    Brand2x12    Brand 2 * x1 2     3.52737     1      1.87813
             7    Brand3x11    Brand 3 * x1 1     3.64162     1      1.90830
             8    Brand3x12    Brand 3 * x1 2     3.63366     1      1.90622
             9    Brand1x21    Brand 1 * x2 1     2.83014     1      1.68230
            10    Brand1x22    Brand 1 * x2 2     2.99478     1      1.73054
            11    Brand2x21    Brand 2 * x2 1     3.63553     1      1.90671
            12    Brand2x22    Brand 2 * x2 2     3.28224     1      1.81169
            13    Brand3x21    Brand 3 * x2 1     2.94706     1      1.71670
            14    Brand3x22    Brand 3 * x2 2     3.27390     1      1.80939
            15    Brand1x31    Brand 1 * x3 1     3.34840     1      1.82986
            16    Brand1x32    Brand 1 * x3 2     2.60489     1      1.61397
            17    Brand2x31    Brand 2 * x3 1     2.67943     1      1.63690
            18    Brand2x32    Brand 2 * x3 2     3.92194     1      1.98039
            19    Brand3x31    Brand 3 * x3 1     2.72245     1      1.64998
            20    Brand3x32    Brand 3 * x3 2     3.21863     1      1.79405
                                                             ==
                                                             20
```

Relative to the unconstrained design, the design with constraints in the range of three to five is $100 \times 0.5163/0.5099 = 101\%$ efficient. Relative to the unconstrained design, the design with constraints of four is $100 \times 0.3688/0.5099 = 72\%$ efficient. With more iterations (not shown), these numbers become: $100 \times 0.5172/0.5191 = 99.6\%$ efficient and $100 \times 0.4454/0.5191 = 85.8\%$ efficient.

Brand Effects and the Alternative-Swapping Algorithm

This next example has brand effects and uses the alternative-swapping algorithm. It also illustrates properties of the standardized orthogonal contrast coding and relative *D*-efficiency when a 100% efficient design does not exist. The following steps make and display a candidate set of branded alternatives:

```
%mktex(3 ** 4, n=3**4)

%mktlab(data=design, vars=x1-x3 Brand)

data full(drop=i);
   set final;
   array f[3];
   do i = 1 to 3; f[i] = (brand eq i); end;
   run;

proc print data=full(obs=9); run;
```

The `%MktEx` macro makes the linear candidate design. The `%MktLab` macro changes the name of the variable `x4` to `Brand` while retaining the original names for `x1-x3` and original levels (1, 2, 3) for all factors. The DATA step creates the flags. The flag variable, `f1`, flags brand 1 candidates as available for the first alternative. Similarly, `f2` flags brand 2 candidates as available for the second alternative, and so on. The Boolean expression `(brand eq i)` evaluates to 1 if true and 0 if false.

The first part of the candidate set is as follows:

```
 Obs    x1    x2    x3    Brand    f1    f2    f3

   1     1     1     1      1       1     0     0
   2     1     1     1      2       0     1     0
   3     1     1     1      3       0     0     1
   4     1     1     2      1       1     0     0
   5     1     1     2      2       0     1     0
   6     1     1     2      3       0     0     1
   7     1     1     3      1       1     0     0
   8     1     1     3      2       0     1     0
   9     1     1     3      3       0     0     1
```

Notice that the candidate set consists of branded alternatives with flags such that only brand *i* is considered for the *ith* alternative of each choice set.

The following `%ChoicEff` macro step makes the choice design from the candidate set of alternatives:

```
%choiceff(data=full,                /* candidate set of alternatives        */
                                    /* alternative-specific effects model   */
                                    /* zero=' ' no reference level for brand*/
                                    /* brand*x1 ... interactions            */
          model=class(brand brand*x1 brand*x2 brand*x3 / zero=' ' sta),
          nsets=15,                 /* number of choice sets                */
          flags=f1-f3,              /* flag which alt can go where, 3 alts  */
          seed=151,                 /* random number seed                   */
          converge=1e-12,           /* convergence criterion                */
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

The `model=` specification states that `Brand` and `x1-x3` are classification or categorical variables and brand effects and brand by attribute interactions (which are also known as alternative-specific effects, see page 386) are desired. The `zero=' '` specification is like `zero=none` except `zero=none` applies to all factors in the specification whereas `zero=' '` applies to just the first. See page 78 for more information about the `zero=` option. This `zero=' '` specification specifies that there is no reference level for the first factor (`Brand`), and the last level will by default be the reference category for the other factors (`x1-x3`). Hence, the interactions are derived from indicator variables created for all three brands, but only two coded variables for the 3 three-level attributes. We need to do this because we need the alternative-specific effects for all brands, including Brand 3. A standardized orthogonal contrast coding is used for `x1-x3` but not for `Brand` (which uses less-than-full-rank indicators).

Some of the results are as follows:

```
Design   Iteration  D-Efficiency       D-Error
----------------------------------------------
     1       0              0                .
             1              0                .
                      1.23291 (Ridged)
             2              0                .
                      1.24083 (Ridged)
             3              0                .
                      1.24689 (Ridged)
             4              0                .
                      1.25318 (Ridged)
             5              0                .
                      1.25318 (Ridged)

Design   Iteration  D-Efficiency       D-Error
----------------------------------------------
     2       0              0                .
             1              0                .
                      1.21367 (Ridged)
             2              0                .
                      1.24462 (Ridged)
             3              0                .
                      1.24565 (Ridged)
             4              0                .
                      1.24708 (Ridged)
             5              0                .
                      1.24738 (Ridged)
             6              0                .
                      1.25210 (Ridged)
             7              0                .
                      1.25210 (Ridged)


                Final Results

          Design                  1
          Choice Sets            15
          Alternatives            3
          Parameters             20
          Maximum Parameters     30
          D-Efficiency            0
          D-Error                 .
```

```
         Variable                                                Standard
  n      Name            Label                 Variance    DF      Error

  1      Brand1          Brand 1                0.42191     1     0.64955
  2      Brand2          Brand 2                0.42147     1     0.64921
  3      Brand3          Brand 3                 .          0      .
  4      Brand1x11       Brand 1 * x1 1         0.33232     1     0.57648
  5      Brand1x12       Brand 1 * x1 2         0.38822     1     0.62307
  6      Brand2x11       Brand 2 * x1 1         0.30106     1     0.54869
  7      Brand2x12       Brand 2 * x1 2         0.39711     1     0.63017
  8      Brand3x11       Brand 3 * x1 1         0.35380     1     0.59481
  9      Brand3x12       Brand 3 * x1 2         0.37744     1     0.61436
 10      Brand1x21       Brand 1 * x2 1         0.39729     1     0.63031
 11      Brand1x22       Brand 1 * x2 2         0.32450     1     0.56965
 12      Brand2x21       Brand 2 * x2 1         0.38070     1     0.61701
 13      Brand2x22       Brand 2 * x2 2         0.35623     1     0.59685
 14      Brand3x21       Brand 3 * x2 1         0.36905     1     0.60749
 15      Brand3x22       Brand 3 * x2 2         0.34511     1     0.58746
 16      Brand1x31       Brand 1 * x3 1         0.39903     1     0.63169
 17      Brand1x32       Brand 1 * x3 2         0.32132     1     0.56685
 18      Brand2x31       Brand 2 * x3 1         0.42616     1     0.65281
 19      Brand2x32       Brand 2 * x3 2         0.32347     1     0.56874
 20      Brand3x31       Brand 3 * x3 1         0.38295     1     0.61883
 21      Brand3x32       Brand 3 * x3 2         0.34997     1     0.59158
                                                           ==
                                                           20
```

The following list is displayed in the log:

```
Redundant Variables:

Brand3
```

Notice that at each step, the efficiency is zero, but a nonzero ridged value is displayed. This model contains a structural-zero coefficient in `Brand3`. While we need alternative-specific effects for Brand 3 (like `Brand3x11` and `Brand3x12`), we do not need the Brand 3 effect (`Brand3`). This can be seen from both the redundant variables list and from looking at the variance and *df* table. The inclusion of the `Brand3` term in the model makes the efficiency of the design zero. However, the `%ChoicEff` macro can still optimize the goodness of the design by optimizing a ridged efficiency criterion—a small constant is added to each diagonal entry of the information matrix to make it nonsingular. That is what is shown in the iteration history. Unlike the `%MktEx` macro, the `%ChoicEff` macro does not have an explicit `ridge=` option. It automatically ridges, but only when needed. We specify `converge=1e-12` because for this example, iteration stops prematurely with the default convergence criterion.

The following step switches to a full-rank coding, dropping the redundant variable Brand3, and using the output from the last step as the initial design:

```
%choiceff(data=full,                /* candidate set of alternatives       */
          init=best(keep=index),    /* select these alts from candidates   */
                                    /* alternative-specific effects model   */
                                    /* zero=' ' no reference level for brand*/
                                    /* brand*x1 ... interactions            */
          model=class(brand brand*x1 brand*x2 brand*x3 / zero=' ' sta),
          drop=brand3,              /* extra model terms to drop from model */
          seed=522,                 /* random number seed                   */
          nsets=15,                 /* number of choice sets                */
          flags=f1-f3,              /* flag which alt can go where, 3 alts  */
          converge=1e-12,           /* convergence criterion                */
          options=relative,         /* display relative D-efficiency        */
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

The option drop=brand3 is used to drop the parameter with the zero coefficient. We could have moved the brand specification into its own class specification (separate from the alternative-specific effects) and not specified zero=' ' with it (see, for example, page 878). However, sometimes it is easier to specify a model with more terms than you really need, and then list the terms to drop, so that is what we illustrate here. See page 78 for more information about the zero= option.

In this usage of init= with alternative swapping, the only part of the initial design that is required is the Index variable. It contains indices into the candidate set of the alternatives that are used to make the initial design. This method can be used in the situation where the initial design was output from the %ChoiceEff macro. The results are as follows:

```
                    Design   Iteration  D-Efficiency      D-Error
                    ----------------------------------------------
                         1       0          3.01341 *     0.33185
                                 1          3.01341       0.33185

                                  Final Results

                              Design                 1
                              Choice Sets           15
                              Alternatives           3
                              Parameters            20
                              Maximum Parameters    30
                              D-Efficiency      3.0134
                              Relative D-Eff   20.0894
                              D-Error           0.3318
                              1 / Choice Sets   0.0667

              Variable                                              Standard
         n    Name          Label                Variance    DF       Error

         1    Brand1        Brand 1               0.42191     1     0.64955
         2    Brand2        Brand 2               0.42147     1     0.64921
```

```
   3    Brand1x11    Brand 1 * x1 1    0.33232     1     0.57648
   4    Brand1x12    Brand 1 * x1 2    0.38822     1     0.62307
   5    Brand2x11    Brand 2 * x1 1    0.30106     1     0.54869
   6    Brand2x12    Brand 2 * x1 2    0.39711     1     0.63017
   7    Brand3x11    Brand 3 * x1 1    0.35380     1     0.59481
   8    Brand3x12    Brand 3 * x1 2    0.37744     1     0.61436
   9    Brand1x21    Brand 1 * x2 1    0.39729     1     0.63031
  10    Brand1x22    Brand 1 * x2 2    0.32450     1     0.56965
  11    Brand2x21    Brand 2 * x2 1    0.38070     1     0.61701
  12    Brand2x22    Brand 2 * x2 2    0.35623     1     0.59685
  13    Brand3x21    Brand 3 * x2 1    0.36905     1     0.60749
  14    Brand3x22    Brand 3 * x2 2    0.34511     1     0.58746
  15    Brand1x31    Brand 1 * x3 1    0.39903     1     0.63169
  16    Brand1x32    Brand 1 * x3 2    0.32132     1     0.56685
  17    Brand2x31    Brand 2 * x3 1    0.42616     1     0.65281
  18    Brand2x32    Brand 2 * x3 2    0.32347     1     0.56874
  19    Brand3x31    Brand 3 * x3 1    0.38295     1     0.61883
  20    Brand3x32    Brand 3 * x3 2    0.34997     1     0.59158
                                                  ==
                                                  20
```

Notice that now there are no zero parameters so *D*-efficiency can be directly computed. In the preceding step, relative *D*-efficiency was requested, and the value is nowhere near 100. This is because the standardized orthogonal contrast coding was not used for all of the attributes, so relative *D*-efficiency is not on a 0 to 100 scale. It is also interesting to perform the evaluation one more time—this time with the standardized orthogonal contrast coding for brand and the other attributes. This lets us drop the `drop=` option. The following step evaluates the design:

```
%choiceff(data=full,                /* candidate set of alternatives        */
          init=best(keep=index),    /* select these alts from candidates    */
                                    /* alternative-specific effects model   */
                                    /* zero=' ' no reference level for brand*/
                                    /* brand*x1 ... interactions            */
          model=class(brand x1 x2 x3 brand*x1 brand*x2 brand*x3 / sta),
          seed=522,                 /* random number seed                   */
          nsets=15,                 /* number of choice sets                */
          flags=f1-f3,              /* flag which alt can go where, 3 alts  */
          converge=1e-12,           /* convergence criterion                */
          options=relative,         /* display relative D-efficiency        */
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

It is instructive to compare the two `model=` options from the previous evaluation and the current one. They are as follows:

```
model=class(brand          brand*x1 brand*x2 brand*x3 / zero=' ' sta),
model=class(brand x1 x2 x3 brand*x1 brand*x2 brand*x3 /          sta),
```

Previously, there was a brand effect (3 − 1 = 2 parameters) and attribute effects within each of the brands (3 brands times 3 attributes times (3 levels minus 1) = 18 parameters) for a total of 20 parameters. Now there is a brand effect (3 − 1 = 2 parameters), attribute effects (3 attributes times (3 levels minus 1) = 6 parameters), and ((3 brands minus 1) times 3 attributes times (3 levels minus 1) = 12 parameters) for a total of 20 parameters. The number of parameters has not changed, but the names and interpretation has changed. The results are as follows:

```
                              Final Results

                          Design                   1
                          Choice Sets             15
                          Alternatives             3
                          Parameters              20
                          Maximum Parameters      30
                          D-Efficiency        9.5507
                          Relative D-Eff     63.6715
                          D-Error             0.1047
                          1 / Choice Sets     0.0667

         Variable                                                   Standard
    n    Name          Label                Variance     DF          Error

    1    Brand1        Brand 1               0.07032      1        0.26518
    2    Brand2        Brand 2               0.07232      1        0.26892
    3    x11           x1 1                  0.11182      1        0.33440
    4    x12           x1 2                  0.13844      1        0.37208
    5    x21           x2 1                  0.13302      1        0.36472
    6    x22           x2 2                  0.10224      1        0.31975
    7    x31           x3 1                  0.10243      1        0.32005
    8    x32           x3 2                  0.13957      1        0.37360
    9    Brand1x11     Brand 1 * x1 1        0.11015      1        0.33188
   10    Brand1x12     Brand 1 * x1 2        0.12050      1        0.34713
   11    Brand2x11     Brand 2 * x1 1        0.10709      1        0.32725
   12    Brand2x12     Brand 2 * x1 2        0.12865      1        0.35867
   13    Brand1x21     Brand 1 * x2 1        0.13216      1        0.36354
   14    Brand1x22     Brand 1 * x2 2        0.10969      1        0.33119
   15    Brand2x21     Brand 2 * x2 1        0.11716      1        0.34229
   16    Brand2x22     Brand 2 * x2 2        0.13002      1        0.36058
   17    Brand1x31     Brand 1 * x3 1        0.15565      1        0.39452
   18    Brand1x32     Brand 1 * x3 2        0.09699      1        0.31142
   19    Brand2x31     Brand 2 * x3 1        0.14463      1        0.38031
   20    Brand2x32     Brand 2 * x3 2        0.09503      1        0.30826
                                                         ==
                                                         20
```

While these two different codings are equivalent, the former does not use the standardized orthogonal contrast coding for brand, while the latter does. Now, the variances are closer to the hypothetical minimum of one over the number of choice sets, and relative *D*-efficiency is larger, but it is still not close to 100. To understand why, imagine that we were going to construct this design directly from an orthogonal array. We would need a design in 45 runs with a fifteen-level factor, for the choice set number, and 4 three-level factors (`Brand x1-x3`). We would additionally need to estimate the interactions between `Brand` and each of `x1-x3`. Such a design does not exist. We can see this by using the `%MktRuns` macro as follows:

```
%mktruns(15 3 3 3 3, interact=2*3 2*4 2*5)
```

The output of this macro (not shown) shows us that the smallest design in which this could possibly work is 135 runs. It is important to note, however, that unless it reports that it will work in 45 runs, it will not work. That is, we are looking for a design with 15 sets and 3 alternatives, and hence 45 runs. With 135 runs, you would have to see if a design with 45 choice sets worked. Now, returning to the 15 sets and 3 alternatives, the relative *D*-efficiency *is* on a 0 to 100 scale, but it is relative to a *hypothetical* optimal design that cannot possibly exist. This is often the case in both linear and choice modeling. Incidentally, for a main effects only model, an optimal design can be constructed as follows:

```
%mktex(15 3 ** 4, n=45)

%mktlab(data=design, vars=Set Brand x1-x3)

%choiceff(data=final,              /* candidate set of choice sets         */
          init=final(keep=set),     /* select these sets from candidates    */
          model=class(brand x1 x2 x3 / sta), /* model w stdzd orthog coding */
          nsets=15,                 /* 6 choice sets                        */
          nalts=3,                  /* 3 alternatives per set               */
          options=relative,         /* display relative D-efficiency        */
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

Some of the results are as follows:

```
                              Final Results

                        Design                  1
                        Choice Sets            15
                        Alternatives            3
                        Parameters              8
                        Maximum Parameters     30
                        D-Efficiency      15.0000
                        Relative D-Eff   100.0000
                        D-Error            0.0667
                        1 / Choice Sets    0.0667
```

```
           Variable                                        Standard
     n       Name       Label       Variance    DF          Error

     1      Brand1      Brand 1     0.066667     1        0.25820
     2      Brand2      Brand 2     0.066667     1        0.25820
     3      x11         x1 1        0.066667     1        0.25820
     4      x12         x1 2        0.066667     1        0.25820
     5      x21         x2 1        0.066667     1        0.25820
     6      x22         x2 2        0.066667     1        0.25820
     7      x31         x3 1        0.066667     1        0.25820
     8      x32         x3 2        0.066667     1        0.25820
                                                ==
                                                 8
```

The `%ChoicEff` macro can also find this design by searching the full-factorial candidate set, but it takes a while. It comes very close (in the neighborhood of 99% relative *D*-efficiency) very easily, but it has a hard time finding the exact optimal main-effect design for this problem. The relative *D*-efficiency calculations for the alternative-specific effects model are again based on the assumption that a design with a variance structure like this main-effects model exists for the alternative-specific effects model. It has no way of knowing what the optimal variance structure is for models like this where a "perfect" design does not exist.

Alternative-Specific Effects and the Alternative-Swapping Algorithm

This example is provided to show how you can use the `%ChoicEff` macro search for an efficient design for a model with alternative-specific effects. This example is based on the vacation example starting on page 339 in the "Discrete Choice" chapter. That example uses the linear arrangement of a choice design approach to construct a choice design from a near orthogonal array. The approach illustrated in the vacation example starting on page 339 is probably the optimal approach for this problem. However, the approach that is used here works almost as well. When you become a sophisticated designer of choice experiments, you will want to be facile with all of the tools in the discrete choice chapter, the experimental design chapter, and this chapter. However, when you are just starting, you might find it easer to concentrate on simply using the `%ChoicEff` macro with a candidate set of alternatives that you create by using the `%MktEx` macro.

In this example, a researcher is interested in studying choice of vacation destinations. There are five destinations (alternatives) of interest: Hawaii, Alaska, Mexico, California, and Maine. Each alternative is composed of three factors: package cost ($999, $1,249, $1,499), scenery (mountains, lake, beach), and accommodations (cabin, bed & breakfast, and hotel). In addition, there is a stay at home alternative. See page 339 for more information.

The following step creates formats for each of the destination attributes:

```
proc format;
   value price 1 = ' 999'      2 = '1249'            3 = '1499';
   value scene 1 = 'Mountains' 2 = 'Lake'            3 = 'Beach';
   value lodge 1 = 'Cabin'     2 = 'Bed & Breakfast' 3 = 'Hotel';
   run;
```

Since there are three attributes for each alternative, the following step creates a full-factorial design that consists of all of the combinations of levels that can occur for each destination:

```
%mktex(3 ** 3, n=27)
```

The results of this step are not shown, but the full-factorial design is 100% D-efficient, it has $3 \times 3 \times 3 = 27$ runs, and it is stored in the SAS data set `Design`.

The following step creates a candidate set of alternatives:

```
data cand;
   retain f1-f6 0;
   length Place $ 10 Price Scene Lodge 8;
   if _n_ = 1 then do; f6 = 1; Place = 'Home'; output; f6 = 0; end;
   set design(rename=(x1=Price x2=Scene x3=Lodge));
   price = input(put(price, price.), 5.);
   f1 = 1; Place = 'Hawaii';     output; f1 = 0;
   f2 = 1; Place = 'Alaska';     output; f2 = 0;
   f3 = 1; Place = 'Mexico';     output; f3 = 0;
   f4 = 1; Place = 'California'; output; f4 = 0;
   f5 = 1; Place = 'Maine';      output; f5 = 0;
   format scene scene. lodge lodge.;
   run;
```

Each of the 27 runs in the full-factorial design is read in once and written out 5 times, once for each destination. In addition, a constant alternative is written once for a total of $5 \times 27 + 1 = 136$ candidates. Some of the candidates are as follows:

```
f1  f2  f3  f4  f5  f6  Place       Price    Scene         Lodge

 0   0   0   0   0   1  Home           .          .                 .
 1   0   0   0   0   0  Hawaii       999  Mountains  Cabin
 0   1   0   0   0   0  Alaska       999  Mountains  Cabin
 0   0   1   0   0   0  Mexico       999  Mountains  Cabin
 0   0   0   1   0   0  California   999  Mountains  Cabin
 0   0   0   0   1   0  Maine        999  Mountains  Cabin
 1   0   0   0   0   0  Hawaii       999  Mountains  Bed & Breakfast
 0   1   0   0   0   0  Alaska       999  Mountains  Bed & Breakfast
 0   0   1   0   0   0  Mexico       999  Mountains  Bed & Breakfast
 0   0   0   1   0   0  California   999  Mountains  Bed & Breakfast
 0   0   0   0   1   0  Maine        999  Mountains  Bed & Breakfast
 1   0   0   0   0   0  Hawaii       999  Mountains  Hotel
 0   1   0   0   0   0  Alaska       999  Mountains  Hotel
 0   0   1   0   0   0  Mexico       999  Mountains  Hotel
 0   0   0   1   0   0  California   999  Mountains  Hotel
 0   0   0   0   1   0  Maine        999  Mountains  Hotel
 .
 .
 .
 1   0   0   0   0   0  Hawaii        1499  Lake        Cabin
 0   1   0   0   0   0  Alaska        1499  Lake        Cabin
 0   0   1   0   0   0  Mexico        1499  Lake        Cabin
 0   0   0   1   0   0  California    1499  Lake        Cabin
 0   0   0   0   1   0  Maine         1499  Lake        Cabin
 .
 .
 .
 1   0   0   0   0   0  Hawaii        1499  Beach       Hotel
 0   1   0   0   0   0  Alaska        1499  Beach       Hotel
 0   0   1   0   0   0  Mexico        1499  Beach       Hotel
 0   0   0   1   0   0  California    1499  Beach       Hotel
 0   0   0   0   1   0  Maine         1499  Beach       Hotel
```

Consider again the data step that creates this candidate set:

```
data cand;
   retain f1-f6 0;
   length Place $ 10 Price Scene Lodge 8;
   if _n_ = 1 then do; f6 = 1; Place = 'Home'; output; f6 = 0; end;
   set design(rename=(x1=Price x2=Scene x3=Lodge));
   price = input(put(price, price.), 5.);
   f1 = 1; Place = 'Hawaii';     output; f1 = 0;
   f2 = 1; Place = 'Alaska';     output; f2 = 0;
   f3 = 1; Place = 'Mexico';     output; f3 = 0;
   f4 = 1; Place = 'California'; output; f4 = 0;
   f5 = 1; Place = 'Maine';      output; f5 = 0;
   format scene scene. lodge lodge.;
   run;
```

The `retain` statement creates the six flag variables, `f1-f6` (one for each alternative), initializes them to zero (this candidate cannot be used for any alternatives), and retains their values so that they are not set to missing at each new pass through the DATA step. The `length` statement specifies that the variable `Place` is a character variable of length 10 and that the remaining variables are numeric. The placement of the `retain` and `length` statements ensures that the flag variables appear first in the data set followed by the destination and attribute variables. This is purely for aesthetic reasons when displaying the candidates and does not affect the design search. The next statement adds a single candidate for the constant alternative (`f6 = 1` and `f1-f5 = 0`). The attributes `Price`, `Scene`, and `Lodge` have missing values since the design has not been read yet. The next statement reads each of the 27 candidates and renames the factor names into attribute names. The next statement maps the price attribute from numeric values of 1, 2, 3 to numeric values of 999, 1249, and 1499 by formatting the numeric value by using the `put` function and then converting the result to numeric by using the `input` function. The next five lines write out a candidate for each of the five nonconstant alternatives. Flag variables are set such that `f1` is 1 and `f2-f4` are 0 for the first alternative, `f2` is 1 and `f1 f3-f4` are 0 for the second alternative, and so on. Finally, formats are assigned. The flag variables control which alternative (or for some designs, which alternatives) each candidate can be used.

The next step searches the candidate set for a design with alternative-specific effects:

```
%choiceff(data=cand,               /* candidate set of alternatives       */
          model=class(place /      /* alternative effects                 */
                      zero=none    /* zero=none - use all levels          */
                      order=data)  /* use ordering of levels from data set */
                class(place * price /* alternative-specific effect of price */
                      place * scene /* alternative-specific effect of scene */
                      place * lodge /* alternative-specific effect of lodge */
                    / zero=none    /* zero=none - use all levels of place  */
                      order=formatted)/* order=formatted - sort levels     */
              / lprefix=0          /* lpr=0 labels created from just levels*/
                cprefix=0          /* cpr=0 names created from just levels */
                separators=' ' ', ',/* use comma sep to build interact terms*/
          nsets=36,                /* number of choice sets               */
          flags=f1-f6,             /* six alternatives, alt-specific      */
          seed=104,                /* random number seed                  */
          beta=zero)               /* assumed beta vector, Ho: b=0        */
```

As we often do, we ask for every conceivable parameter during our first pass, including those that are structural zeros, which we will eliminate in a subsequent pass. Some of the results are as follows:

```
                   Design   Iteration  D-Efficiency      D-Error
                   ----------------------------------------------
                        1       0                 0            .
                                1                 0            .
                                         0.00072512 (Ridged)
                                2                 0            .
                                         0.00072588 (Ridged)

                   Design   Iteration  D-Efficiency      D-Error
                   ----------------------------------------------
                        2       0                 0            .
                                1                 0            .
                                         0.00072537 (Ridged)
                                2                 0            .
                                         0.00072647 (Ridged)

Redundant Variables:

Maine Alaska_1499 California_1499 Hawaii_1499 Home_999 Home_1249 Home_1499
Maine_1499 Mexico_1499 AlaskaMountains CaliforniaMountains HawaiiMountains
HomeBeach HomeLake HomeMountains MaineMountains MexicoMountains AlaskaHotel
CaliforniaHotel HawaiiHotel HomeBed___Breakfast HomeCabin HomeHotel MaineHotel
MexicoHotel
```

```
                    Final Results

              Design                   1
              Choice Sets             36
              Alternatives             6
              Parameters              35
              Maximum Parameters     180
              D-Efficiency             0
              D-Error                  .


                                                                        Standard
 n Variable Name            Label                        Variance DF     Error

 1 Home                     Home                          1.56505  1   1.25102
 2 Hawaii                   Hawaii                        2.73192  1   1.65285
 3 Alaska                   Alaska                        2.92946  1   1.71157
 4 Mexico                   Mexico                        2.63759  1   1.62407
 5 California               California                    2.70000  1   1.64317
 6 Maine                    Maine                          .       0    .
 7 Alaska_999               Alaska,  999                  1.22388  1   1.10629
 8 Alaska_1249              Alaska, 1249                  1.23861  1   1.11293
 9 Alaska_1499              Alaska, 1499                   .       0    .
10 California_999           California,  999              1.23793  1   1.11262
11 California_1249          California, 1249              1.21703  1   1.10319
12 California_1499          California, 1499               .       0    .
13 Hawaii_999               Hawaii,  999                  1.22456  1   1.10660
14 Hawaii_1249              Hawaii, 1249                  1.22929  1   1.10873
15 Hawaii_1499              Hawaii, 1499                   .       0    .
16 Home_999                 Home,  999                     .       0    .
17 Home_1249                Home, 1249                     .       0    .
18 Home_1499                Home, 1499                     .       0    .
19 Maine_999                Maine,  999                   1.23864  1   1.11294
20 Maine_1249               Maine, 1249                   1.22918  1   1.10868
21 Maine_1499               Maine, 1499                    .       0    .
22 Mexico_999               Mexico,  999                  1.23168  1   1.10981
23 Mexico_1249              Mexico, 1249                  1.22689  1   1.10765
24 Mexico_1499              Mexico, 1499                   .       0    .
25 AlaskaBeach              Alaska, Beach                 1.22235  1   1.10560
26 AlaskaLake               Alaska, Lake                  1.22247  1   1.10565
27 AlaskaMountains          Alaska, Mountains              .       0    .
28 CaliforniaBeach          California, Beach             1.24270  1   1.11477
29 CaliforniaLake           California, Lake              1.25330  1   1.11951
30 CaliforniaMountains      California, Mountains          .       0    .
31 HawaiiBeach              Hawaii, Beach                 1.22975  1   1.10894
32 HawaiiLake               Hawaii, Lake                  1.22507  1   1.10683
33 HawaiiMountains          Hawaii, Mountains              .       0    .
34 HomeBeach                Home, Beach                    .       0    .
35 HomeLake                 Home, Lake                     .       0    .
36 HomeMountains            Home, Mountains                .       0    .
```

```
     37 MaineBeach                Maine, Beach                  1.23600  1  1.11175
     38 MaineLake                 Maine, Lake                   1.23112  1  1.10956
     39 MaineMountains            Maine, Mountains               .       0   .
     40 MexicoBeach               Mexico, Beach                 1.23252  1  1.11019
     41 MexicoLake                Mexico, Lake                  1.22701  1  1.10771
     42 MexicoMountains           Mexico, Mountains              .       0   .
     43 AlaskaBed___Breakfast     Alaska, Bed & Breakfast       1.23723  1  1.11231
     44 AlaskaCabin               Alaska, Cabin                 1.22093  1  1.10496
     45 AlaskaHotel               Alaska, Hotel                  .       0   .
     46 CaliforniaBed___Breakfast California, Bed & Breakfast   1.23418  1  1.11094
     47 CaliforniaCabin           California, Cabin             1.22615  1  1.10732
     48 CaliforniaHotel           California, Hotel              .       0   .
     49 HawaiiBed___Breakfast     Hawaii, Bed & Breakfast       1.22431  1  1.10648
     50 HawaiiCabin               Hawaii, Cabin                 1.23680  1  1.11211
     51 HawaiiHotel               Hawaii, Hotel                  .       0   .
     52 HomeBed___Breakfast       Home, Bed & Breakfast          .       0   .
     53 HomeCabin                 Home, Cabin                    .       0   .
     54 HomeHotel                 Home, Hotel                    .       0   .
     55 MaineBed___Breakfast      Maine, Bed & Breakfast        1.22375  1  1.10623
     56 MaineCabin                Maine, Cabin                  1.22955  1  1.10885
     57 MaineHotel                Maine, Hotel                   .       0   .
     58 MexicoBed___Breakfast     Mexico, Bed & Breakfast       1.23665  1  1.11205
     59 MexicoCabin               Mexico, Cabin                 1.23940  1  1.11328
     60 MexicoHotel               Mexico, Hotel                  .       0   .
                                                                        ==
                                                                        35
```

Since there are extra parameters, the *D*-efficiency is zero, but the `%ChoicEff` macro optimizes a ridged efficiency criterion. A list of reference parameters is provided. These can be used to drop the extra parameters, or we can use the `zero=` option. There is one reference level for destination and one for each of the nonconstant destination attributes. Furthermore, all of the parameters associated with the constant stay at home alternative are zero.[†] There are a total of 35 parameters that can be estimated (5 destinations plus 5 destinations times 3 attributes times $(3-1)$ levels. The model specification `model=class(place / zero=none order=data) class(place * price place * scene place * lodge / zero=none order=formatted)` creates one term for each destination (minus one) and two terms for each destination and attribute combination.

The standardized orthogonal contrast coding is not used since we have five nonconstant alternatives and three-level factors. It will be hard to know the optimum *D*-efficiency for a design like this.

We can run this again designating the stay at home level as a reference level and ask for 100 designs this time as follows:

[†]When we more explicitly control the reference level, Maine will have an estimable parameter for the destination effect instead of the stay at home alternative.

```
   %choiceff(data=cand,                /* candidate set of alternatives       */
             model=class(place /       /* alternative effects                 */
                         zero='Home'   /* use 'Home' as reference level       */
                         order=data)   /* use ordering of levels from data set */
                                       /* place * price ... - interactions or  */
                   class(place * price /* alternative-specific effect of price */
                         place * scene /* alternative-specific effect of scene */
                         place * lodge /* alternative-specific effect of lodge */
                       / zero='Home'   /* use 'Home' as reference level       */
                         order=formatted)/* order=formatted - sort levels      */
                  / lprefix=0          /* lpr=0 labels created from just levels*/
                    cprefix=0          /* cpr=0 names created from just levels */
                    separators=' ' ', ',/* use comma sep to build interact terms*/
             nsets=36,                 /* number of choice sets               */
             flags=f1-f6,              /* six alternatives, alt-specific      */
             maxiter=100,              /* maximum number of designs to make   */
             seed=104,                 /* random number seed                  */
             beta=zero)                /* assumed beta vector, Ho: b=0        */
```

Some of the results are as follows:

```
Design   Iteration  D-Efficiency      D-Error
----------------------------------------------
     1       0           1.02898 *     0.97183
             1           1.17248 *     0.85290
             2           1.17458 *     0.85137

Design   Iteration  D-Efficiency      D-Error
----------------------------------------------
     2       0           1.05615       0.94684
             1           1.17317       0.85239
             2           1.17621 *     0.85019

Design   Iteration  D-Efficiency      D-Error
----------------------------------------------
     3       0           1.05667       0.94637
             1           1.17486       0.85117
             2           1.17642 *     0.85003

Design   Iteration  D-Efficiency      D-Error
----------------------------------------------
     4       0           1.07743       0.92814
             1           1.17467       0.85130
             2           1.17467       0.85130

Design   Iteration  D-Efficiency      D-Error
----------------------------------------------
     5       0           1.07993       0.92598
             1           1.17571       0.85055
             2           1.17663 *     0.84989
```

```
         .
         .
         .

    Design   Iteration  D-Efficiency      D-Error
    ---------------------------------------------
        53       0           1.04235      0.95937
                 1           1.17464      0.85132
                 2           1.17728 *    0.84942
         .
         .
         .

    Design   Iteration  D-Efficiency      D-Error
    ---------------------------------------------
       100       0           1.04486      0.95706
                 1           1.17310      0.85244
                 2           1.17662      0.84989

                      Final Results

              Design                  53
              Choice Sets             36
              Alternatives             6
              Parameters              35
              Maximum Parameters     180
              D-Efficiency        1.1773
              D-Error             0.8494

 n Variable Name             Label                        Variance DF   Error

 1 Hawaii                    Hawaii                        1.55830  1  1.24832
 2 Alaska                    Alaska                        1.56263  1  1.25005
 3 Mexico                    Mexico                        1.55387  1  1.24654
 4 California                California                    1.55178  1  1.24570
 5 Maine                     Maine                         1.54976  1  1.24489
 6 Alaska_999                Alaska,  999                  1.23707  1  1.11224
 7 Alaska_1249               Alaska, 1249                  1.22530  1  1.10693
 8 California_999            California,  999              1.22238  1  1.10561
 9 California_1249           California, 1249              1.22445  1  1.10655
10 Hawaii_999                Hawaii,  999                  1.21990  1  1.10449
11 Hawaii_1249               Hawaii, 1249                  1.21886  1  1.10402
12 Maine_999                 Maine,  999                   1.22909  1  1.10864
13 Maine_1249                Maine, 1249                   1.21766  1  1.10348
14 Mexico_999                Mexico,  999                  1.22986  1  1.10899
15 Mexico_1249               Mexico, 1249                  1.22239  1  1.10562
16 AlaskaBeach               Alaska, Beach                 1.21993  1  1.10450
17 AlaskaLake                Alaska, Lake                  1.23474  1  1.11119
18 CaliforniaBeach           California, Beach             1.22751  1  1.10793
19 CaliforniaLake            California, Lake              1.22852  1  1.10839
```

```
  20 HawaiiBeach               Hawaii, Beach                1.21878  1  1.10399
  21 HawaiiLake                Hawaii, Lake                 1.23243  1  1.11015
  22 MaineBeach                Maine, Beach                 1.23564  1  1.11159
  23 MaineLake                 Maine, Lake                  1.22895  1  1.10858
  24 MexicoBeach               Mexico, Beach                1.22240  1  1.10562
  25 MexicoLake                Mexico, Lake                 1.21919  1  1.10417
  26 AlaskaBed___Breakfast     Alaska, Bed & Breakfast      1.24157  1  1.11426
  27 AlaskaCabin               Alaska, Cabin                1.22303  1  1.10591
  28 CaliforniaBed___Breakfast California, Bed & Breakfast  1.22586  1  1.10718
  29 CaliforniaCabin           California, Cabin            1.22318  1  1.10597
  30 HawaiiBed___Breakfast     Hawaii, Bed & Breakfast      1.23355  1  1.11065
  31 HawaiiCabin               Hawaii, Cabin                1.22510  1  1.10684
  32 MaineBed___Breakfast      Maine, Bed & Breakfast       1.21813  1  1.10369
  33 MaineCabin                Maine, Cabin                 1.21869  1  1.10394
  34 MexicoBed___Breakfast     Mexico, Bed & Breakfast      1.22010  1  1.10458
  35 MexicoCabin               Mexico, Cabin                1.22302  1  1.10590
                                                                    ==
                                                                    35
```

Now, *D*-efficiency is not zero and all 35 parameters can be estimated. Note that the stay at home alternative is now consistently the reference level. The *D*-efficiency and variances are similar to those found in the example starting on page 339. However, the efficiency is a bit lower, and the variances are a bit larger. The approach outlined on 339 is a better approach for this problem, but this approach works quite well for this problem and a wide variety of other problems.

The design is displayed in the following step:

```
options ps=200 missing=' ';
proc print;
   id set;
   by set;
   var place -- lodge;
   run;
options ps=60 missing='.';
```

Some of the results are as follows:

```
             Set    Place         Price    Scene        Lodge

               1    Hawaii         1499    Beach        Hotel
                    Alaska          999    Beach        Bed & Breakfast
                    Mexico         1499    Lake         Hotel
                    California     1499    Mountains    Cabin
                    Maine          1249    Mountains    Hotel
                    Home
```

```
  2    Hawaii          1499    Lake           Cabin
       Alaska          1499    Lake           Hotel
       Mexico           999    Mountains      Bed & Breakfast
       California      1249    Beach          Bed & Breakfast
       Maine           1249    Lake           Hotel
       Home

  3    Hawaii          1499    Lake           Bed & Breakfast
       Alaska          1499    Beach          Bed & Breakfast
       Mexico          1499    Lake           Cabin
       California      1249    Lake           Cabin
       Maine            999    Beach          Bed & Breakfast
       Home

  4    Hawaii          1499    Beach          Cabin
       Alaska          1499    Beach          Cabin
       Mexico          1499    Mountains      Bed & Breakfast
       California      1499    Mountains      Bed & Breakfast
       Maine           1249    Mountains      Bed & Breakfast
       Home

  5    Hawaii           999    Mountains      Hotel
       Alaska          1249    Lake           Hotel
       Mexico           999    Lake           Hotel
       California      1499    Lake           Cabin
       Maine           1249    Lake           Cabin
       Home

  6    Hawaii          1499    Lake           Hotel
       Alaska          1249    Lake           Cabin
       Mexico          1499    Lake           Cabin
       California       999    Beach          Hotel
       Maine           1499    Lake           Hotel
       Home
  .
  .
  .
 34    Hawaii          1499    Mountains      Bed & Breakfast
       Alaska          1249    Beach          Hotel
       Mexico           999    Mountains      Bed & Breakfast
       California      1499    Beach          Cabin
       Maine            999    Beach          Hotel
       Home

 35    Hawaii          1249    Mountains      Hotel
       Alaska          1499    Lake           Bed & Breakfast
       Mexico          1249    Beach          Bed & Breakfast
       California      1249    Mountains      Hotel
       Maine           1249    Beach          Hotel
       Home
```

```
              36    Hawaii        1249    Mountains    Cabin
                    Alaska        1499    Beach        Cabin
                    Mexico        1249    Lake         Bed & Breakfast
                    California    1249    Lake         Bed & Breakfast
                    Maine         1499    Mountains    Cabin
                    Home
```

Note that each destination always appears in the same alternative, and this was controlled at candidate set creation time.

The following step codes the design:

```
proc transreg data=best design norestoremissing;
   model class(place / zero='Home' order=data)
         class(place * price place * scene place * lodge /
               zero='Home' order=formatted) /
         lprefix=0 cprefix=0 separators=' ' ', ';
   output out=coded;
   run;
```

The `design` option specifies that no model is fit; the procedure is just being used to code a design. When `design` is specified, dependent variables are not required. The `norestoremissing` option specifies that missing values should not be restored when the `out=` data set is created. By default, the coded `class` variable contains a row of missing values for observations in which the `class` variable is missing. With the `norestoremissing` option, these observations contain a row of zeros instead. This option is useful when there is a constant alternative indicated by missing values.

The following steps display the first coded choice set:

```
proc print data=coded(obs=6) noobs label;
   var place -- lodge Hawaii -- Maine;
   run;

proc print data=coded(obs=6) noobs label;
   var place Alaska_999 Alaska_1249 California_999 California_1249;
   run;

proc print data=coded(obs=6) noobs label;
   var place Hawaii_999 Hawaii_1249 Maine_999 Maine_1249 Mexico_999 Mexico_1249;
   run;

proc print data=coded(obs=6) noobs label;
   var place AlaskaBeach AlaskaLake CaliforniaBeach CaliforniaLake;
   run;

proc print data=coded(obs=6) noobs label;
   var place HawaiiBeach HawaiiLake MaineBeach MaineLake MexicoBeach MexicoLake;
   run;
```

```
proc print data=coded(obs=6) noobs label;
   var place AlaskaBed___Breakfast AlaskaCabin
       CaliforniaBed___Breakfast CaliforniaCabin;
   run;

proc print data=coded(obs=6) noobs label;
   var place HawaiiBed___Breakfast HawaiiCabin MaineBed___Breakfast MaineCabin
        MexicoBed___Breakfast MexicoCabin;
   run;
```

The results are as follows:

```
Place      Price   Scene        Lodge      Hawaii Alaska Mexico California Maine

Hawaii      1499 Beach     Hotel              1      0      0         0       0
Alaska       999 Beach     Bed & Breakfast    0      1      0         0       0
Mexico      1499 Lake      Hotel              0      0      1         0       0
California  1499 Mountains Cabin              0      0      0         1       0
Maine       1249 Mountains Hotel              0      0      0         0       1
Home           .         .               .    0      0      0         0       0

                  Alaska,    Alaska,    California,    California,
         Place      999        1249         999            1249

         Hawaii      0          0            0              0
         Alaska      1          0            0              0
         Mexico      0          0            0              0
         California  0          0            0              0
         Maine       0          0            0              0
         Home        0          0            0              0

             Hawaii,    Hawaii,    Maine,    Maine,    Mexico,    Mexico,
   Place       999        1249       999      1249       999        1249

   Hawaii       0          0          0         0         0          0
   Alaska       0          0          0         0         0          0
   Mexico       0          0          0         0         0          0
   California   0          0          0         0         0          0
   Maine        0          0          0         1         0          0
   Home         0          0          0         0         0          0

                  Alaska,    Alaska,    California,    California,
         Place     Beach       Lake        Beach           Lake

         Hawaii      0          0            0              0
         Alaska      1          0            0              0
         Mexico      0          0            0              0
         California  0          0            0              0
         Maine       0          0            0              0
         Home        0          0            0              0
```

```
                   Hawaii,     Hawaii,     Maine,     Maine,     Mexico,     Mexico,
Place               Beach        Lake       Beach      Lake       Beach        Lake

Hawaii                1           0           0          0          0           0
Alaska                0           0           0          0          0           0
Mexico                0           0           0          0          0           1
California            0           0           0          0          0           0
Maine                 0           0           0          0          0           0
Home                  0           0           0          0          0           0

                         Alaska,                California,
                          Bed &       Alaska,      Bed &       California,
     Place              Breakfast      Cabin     Breakfast        Cabin

     Hawaii                 0            0           0              0
     Alaska                 1            0           0              0
     Mexico                 0            0           0              0
     California             0            0           0              1
     Maine                  0            0           0              0
     Home                   0            0           0              0
```

Note that all coded variables for the stay at home alternative are zero. This is true in all other choice sets as well. Hence the utility for that alternative is zero, and the utility for all other alternatives is relative to zero.

Brand Effects and the Choice Set Swapping Algorithm

This example is provided for complete coverage of the `%ChoicEff` macro. If you are just getting started, concentrate instead on examples of the `%ChoicEff` macro that use candidate sets of alternatives. These next steps handle the same problem, only this time, we use the set-swapping algorithm, and we will specify a parameter vector that is not zero. At first, we omit the `beta=` option, just to see the coding. We specify the `effects` option in the PROC TRANSREG `class` specification to get –1, 0, 1 coding. The following steps create the design:

```
%mktex(3 ** 9, n=2187, seed=121)

data key;
   input (Brand x1-x3) ($);
   datalines;
1 x1 x2 x3
2 x4 x5 x6
3 x7 x8 x9
;
```

```
%mktroll(design=design, key=key, alt=brand, out=rolled)

%choiceff(data=rolled,             /* candidate set of choice sets        */
                                   /* alternative-specific model          */
                                   /* effects coding of interactions      */
                                   /* zero=' ' no reference level for brand*/
                                   /* brand*x1 ... interactions           */
          model=class(brand)
                class(brand*x1 brand*x2 brand*x3 / effects zero=' '),
          nsets=15,                /* number of choice sets               */
          nalts=3)                 /* number of alternatives              */
```

The output tells us the parameter names and the order in which we need to specify parameters. The results are as follows:

```
                         n    Name         Beta    Label

                         1    Brand1         .     Brand 1
                         2    Brand2         .     Brand 2
                         3    Brand1x11      .     Brand 1 * x1 1
                         4    Brand1x12      .     Brand 1 * x1 2
                         5    Brand2x11      .     Brand 2 * x1 1
                         6    Brand2x12      .     Brand 2 * x1 2
                         7    Brand3x11      .     Brand 3 * x1 1
                         8    Brand3x12      .     Brand 3 * x1 2
                         9    Brand1x21      .     Brand 1 * x2 1
                        10    Brand1x22      .     Brand 1 * x2 2
                        11    Brand2x21      .     Brand 2 * x2 1
                        12    Brand2x22      .     Brand 2 * x2 2
                        13    Brand3x21      .     Brand 3 * x2 1
                        14    Brand3x22      .     Brand 3 * x2 2
                        15    Brand1x31      .     Brand 1 * x3 1
                        16    Brand1x32      .     Brand 1 * x3 2
                        17    Brand2x31      .     Brand 2 * x3 1
                        18    Brand2x32      .     Brand 2 * x3 2
                        19    Brand3x31      .     Brand 3 * x3 1
                        20    Brand3x32      .     Brand 3 * x3 2
```

Now that we are sure we know the order of the parameters, we can specify the assumed betas in the `beta=` option. These numbers are based on prior research or our expectations of approximately what we expect the parameter estimates will be. We also specify `n=100` in this run, which is a sample size we are considering.

The following step creates the design:

```
   %choiceff(data=rolled,             /* candidate set of choice sets        */
                                      /* alternative-specific model          */
                                      /* effects coding of interactions      */
                                      /* zero=' ' no reference level for brand*/
                                      /* brand*x1 ... interactions           */
             model=class(brand)
                   class(brand*x1 brand*x2 brand*x3 / effects zero=' '),

             nsets=15,                /* number of choice sets               */
             nalts=3,                 /* number of alternatives              */
             n=100,                   /* n obs to use in variance formula    */
             seed=462,                /* random number seed                  */
             beta=1 2 -0.5 0.5 -0.75 0.75 -1 1
                  -0.5 0.5 -0.75 0.75 -1 1 -0.5 0.5 -0.75 0.75 -1 1)
```

Some of the results are as follows:

```
                               Final Results

                          Design                 2
                          Choice Sets           15
                          Alternatives           3
                          Parameters            20
                          Maximum Parameters    30
                          D-Efficiency    144.1951
                          D-Error         0.006935

                                                                       Prob >
       Variable                                Assumed    Standard             Squared
     n Name      Label          Variance        Beta   DF    Error      Wald     Wald

     1 Brand1    Brand 1        0.011889        1.00    1   0.10903    9.1714  0.0001
     2 Brand2    Brand 2        0.020697        2.00    1   0.14386   13.9021  0.0001
     3 Brand1x11 Brand 1 * x1 1 0.008617       -0.50    1   0.09283   -5.3865  0.0001
     4 Brand1x12 Brand 1 * x1 2 0.008527        0.50    1   0.09234    5.4147  0.0001
     5 Brand2x11 Brand 2 * x1 1 0.009283       -0.75    1   0.09635   -7.7842  0.0001
     6 Brand2x12 Brand 2 * x1 2 0.012453        0.75    1   0.11159    6.7208  0.0001
     7 Brand3x11 Brand 3 * x1 1 0.021764       -1.00    1   0.14753   -6.7784  0.0001
     8 Brand3x12 Brand 3 * x1 2 0.015657        1.00    1   0.12513    7.9917  0.0001
     9 Brand1x21 Brand 1 * x2 1 0.012520       -0.50    1   0.11189   -4.4685  0.0001
    10 Brand1x22 Brand 1 * x2 2 0.010685        0.50    1   0.10337    4.8370  0.0001
    11 Brand2x21 Brand 2 * x2 1 0.010545       -0.75    1   0.10269   -7.3035  0.0001
    12 Brand2x22 Brand 2 * x2 2 0.012654        0.75    1   0.11249    6.6672  0.0001
    13 Brand3x21 Brand 3 * x2 1 0.018279       -1.00    1   0.13520   -7.3964  0.0001
    14 Brand3x22 Brand 3 * x2 2 0.012117        1.00    1   0.11008    9.0845  0.0001
    15 Brand1x31 Brand 1 * x3 1 0.009697       -0.50    1   0.09848   -5.0774  0.0001
    16 Brand1x32 Brand 1 * x3 2 0.010787        0.50    1   0.10386    4.8141  0.0001
```

```
17 Brand2x31 Brand 2 * x3 1 0.009203  -0.75   1  0.09593  -7.8181 0.0001
18 Brand2x32 Brand 2 * x3 2 0.013923   0.75   1  0.11800   6.3562 0.0001
19 Brand3x31 Brand 3 * x3 1 0.016546  -1.00   1  0.12863  -7.7742 0.0001
20 Brand3x32 Brand 3 * x3 2 0.014235   1.00   1  0.11931   8.3815 0.0001
                                             ==
                                             20
```

First, notice that D-efficiency is not on a 0 to 100 scale with this design specification. Also notice that parameters and test statistics are incorporated into the output. The n= value is incorporated into the variance matrix and hence the efficiency statistics, variances and tests.

Cross-Effects and the Choice Set Swapping Algorithm

These next steps create a design for a cross-effects model with five brands at three prices and a constant alternative:

```
%mktex(3 ** 5, n=3**5)

data key;
   input (Brand Price) ($);
   datalines;
1 x1
2 x2
3 x3
4 x4
5 x5
. .
;

%mktroll(design=design, key=key, alt=brand, out=rolled, keep=x1-x5)

proc print; by set; id set; where set in (1, 48, 101, 243); run;
```

See the examples beginning on pages 444 and 468 for more information about cross-effects. Note the choice-set-swapping algorithm can handle cross-effects but not the alternative-swapping algorithm.

The keep= option in the %MktRoll macro is used to keep the price variables that are needed to make the cross-effects. The following display shows some of the candidate choice sets:

```
  Set    Brand    Price    x1    x2    x3    x4    x5

    1      1        1       1     1     1     1     1
           2        1       1     1     1     1     1
           3        1       1     1     1     1     1
           4        1       1     1     1     1     1
           5        1       1     1     1     1     1
                    .       1     1     1     1     1
```

```
     48        1        1        1     2     3     1     3
               2        2        1     2     3     1     3
               3        3        1     2     3     1     3
               4        1        1     2     3     1     3
               5        3        1     2     3     1     3
                        .        1     2     3     1     3

    101        1        2        2     1     3     1     2
               2        1        2     1     3     1     2
               3        3        2     1     3     1     2
               4        1        2     1     3     1     2
               5        2        2     1     3     1     2
                        .        2     1     3     1     2

    243        1        3        3     3     3     3     3
               2        3        3     3     3     3     3
               3        3        3     3     3     3     3
               4        3        3     3     3     3     3
               5        3        3     3     3     3     3
                        .        3     3     3     3     3
```

Notice that x1 contains the price for Brand 1, x2 contains the price for Brand 2, and so on, and the price of brand *i* in a choice set is the same, no matter which alternative it is stored with.

The following %ChoicEff macro step creates the choice design with cross-effects:

```
%choiceff(data=rolled,              /* candidate set of choice sets         */
                                    /* model with cross-effects             */
                                    /* zero=none - use all levels           */
                                    /* ide(...) * class(...) - cross-effects*/
          model=class(brand brand*price / zero=none)
                identity(x1-x5) * class(brand / zero=none),
          nsets=20,                 /* number of choice sets                */
          nalts=6,                  /* number of alternatives               */
          seed=17,                  /* random number seed                   */
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

Cross-effects are created by interacting the price factors with brand. See pages 452 and 509 for more information about cross-effects.

The following redundant variable list is displayed in the log:

```
Redundant Variables:

Brand1Price3 Brand2Price3 Brand3Price3 Brand4Price3 Brand5Price3 x1Brand1
x2Brand2 x3Brand3 x4Brand4 x5Brand5
```

Next, we will run the macro again, this time requesting a full-rank model. The list of dropped names was created by copying from the redundant variable list. Also, zero=none was changed to zero=' ' so no level would be zeroed for Brand but the last level of Price would be zeroed. See page 78 for more information about the zero= option.

The following step creates the design:

```
%choiceff(data=rolled,              /* candidate set of choice sets        */
                                    /* zero=' ' no reference level for brand*/
                                    /* model with cross-effects            */
                                    /* zero=none - use all levels          */
                                    /* ide(...) * class(...) - cross-effects*/
          model=class(brand brand*price / zero=' ')
                identity(x1-x5) * class(brand / zero=none),

                                    /* extra model terms to drop from model */
          drop=x1Brand1 x2Brand2 x3Brand3 x4Brand4 x5Brand5,
          nsets=20,                 /* number of choice sets               */
          nalts=6,                  /* number of alternatives              */
          seed=17,                  /* random number seed                  */
          beta=zero)                /* assumed beta vector, Ho: b=0        */
```

In the following results, notice that we have five brand parameters, two price parameters for each of the five brands, and four cross-effect parameters for each of the five brands:

```
         Variable                                               Standard
   n     Name           Label                 Variance    DF       Error

   1     Brand1         Brand 1                13.8149     1     3.71683
   2     Brand2         Brand 2                13.5263     1     3.67782
   3     Brand3         Brand 3                13.2895     1     3.64547
   4     Brand4         Brand 4                13.5224     1     3.67728
   5     Brand5         Brand 5                16.3216     1     4.04000
   6     Brand1Price1   Brand 1 * Price 1       2.8825     1     1.69779
   7     Brand1Price2   Brand 1 * Price 2       3.5118     1     1.87399
   8     Brand2Price1   Brand 2 * Price 1       2.8710     1     1.69441
   9     Brand2Price2   Brand 2 * Price 2       3.5999     1     1.89733
  10     Brand3Price1   Brand 3 * Price 1       2.8713     1     1.69448
  11     Brand3Price2   Brand 3 * Price 2       3.5972     1     1.89662
  12     Brand4Price1   Brand 4 * Price 1       2.8710     1     1.69441
  13     Brand4Price2   Brand 4 * Price 2       3.5560     1     1.88574
  14     Brand5Price1   Brand 5 * Price 1       2.8443     1     1.68649
  15     Brand5Price2   Brand 5 * Price 2       3.8397     1     1.95953
  16     x1Brand2       x1 * Brand 2            0.7204     1     0.84878
  17     x1Brand3       x1 * Brand 3            0.7209     1     0.84908
  18     x1Brand4       x1 * Brand 4            0.7204     1     0.84878
  19     x1Brand5       x1 * Brand 5            0.7204     1     0.84877
  20     x2Brand1       x2 * Brand 1            0.7178     1     0.84722
  21     x2Brand3       x2 * Brand 3            0.7178     1     0.84724
  22     x2Brand4       x2 * Brand 4            0.7178     1     0.84720
  23     x2Brand5       x2 * Brand 5            0.7248     1     0.85133
```

```
      24    x3Brand1         x3 * Brand 1          0.7178     1      0.84722
      25    x3Brand2         x3 * Brand 2          0.7178     1      0.84721
      26    x3Brand4         x3 * Brand 4          0.7178     1      0.84720
      27    x3Brand5         x3 * Brand 5          0.7248     1      0.85133
      28    x4Brand1         x4 * Brand 1          0.7178     1      0.84722
      29    x4Brand2         x4 * Brand 2          0.7178     1      0.84721
      30    x4Brand3         x4 * Brand 3          0.7178     1      0.84724
      31    x4Brand5         x4 * Brand 5          0.7293     1      0.85402
      32    x5Brand1         x5 * Brand 1          0.7111     1      0.84325
      33    x5Brand2         x5 * Brand 2          0.7180     1      0.84737
      34    x5Brand3         x5 * Brand 3          0.7248     1      0.85135
      35    x5Brand4         x5 * Brand 4          0.7179     1      0.84731
                                                             ==
                                                             35
```

Asymmetric Factors and the Alternative Swapping Algorithm

In this `%ChoicEff` macro example, the goal is to create a design for a pricing study with ten brands plus a constant alternative. Each brand has a single attribute, price. However, the prices are potentially different for each brand and they do not even have the same numbers of levels. A model is desired with brand and alternative-specific price effects. The design specifications are as follows:

Brand	Levels	Prices
Brand 1	8	0.89 0.94 0.99 1.04 1.09 1.14 1.19 1.24
Brand 2	8	0.94 0.99 1.04 1.09 1.14 1.19 1.24 1.29
Brand 3	6	0.99 1.04 1.09 1.14 1.19 1.24
Brand 4	6	0.89 0.94 0.99 1.04 1.09 1.14
Brand 5	6	1.04 1.09 1.14 1.19 1.24 1.29
Brand 6	4	0.89 0.99 1.09 1.19
Brand 7	4	0.99 1.09 1.19 1.29
Brand 8	4	0.94 0.99 1.14 1.19
Brand 9	4	1.09 1.14 1.19 1.24
Brand 10	4	1.14 1.19 1.24 1.29

The challenging aspect of this problem is creating the candidate set while coping with the price asymmetries. The candidate set must contain 8 rows for the eight Brand 1 prices, 8 rows for the eight Brand 2 prices, 6 rows for the six Brand 3 prices, ..., and 4 rows for the four Brand 10 prices. It also must contain a constant alternative. Furthermore, if we are to use the alternative-swapping algorithm, the candidate set must contain 11 flag variables, each of which will designate the appropriate group of candidates for each alternative. We could run the `%MktEx` macro ten times to make a candidate set for each of the brands, but since we have only one factor per brand, it would be much easier to generate the candidate set with a DATA step. Before we discuss the code, it is instructive to examine the candidate set.

The candidate set is as follows:

```
 Obs    Brand    Price    f1    f2    f3    f4    f5    f6    f7    f8    f9    f10    f11

   1      1       0.89     1     0     0     0     0     0     0     0     0     0      0
   2      1       0.94     1     0     0     0     0     0     0     0     0     0      0
   3      1       0.99     1     0     0     0     0     0     0     0     0     0      0
   4      1       1.04     1     0     0     0     0     0     0     0     0     0      0
   5      1       1.09     1     0     0     0     0     0     0     0     0     0      0
   6      1       1.14     1     0     0     0     0     0     0     0     0     0      0
   7      1       1.19     1     0     0     0     0     0     0     0     0     0      0
   8      1       1.24     1     0     0     0     0     0     0     0     0     0      0
   9      2       0.94     0     1     0     0     0     0     0     0     0     0      0
  10      2       0.99     0     1     0     0     0     0     0     0     0     0      0
  11      2       1.04     0     1     0     0     0     0     0     0     0     0      0
  12      2       1.09     0     1     0     0     0     0     0     0     0     0      0
  13      2       1.14     0     1     0     0     0     0     0     0     0     0      0
  14      2       1.19     0     1     0     0     0     0     0     0     0     0      0
  15      2       1.24     0     1     0     0     0     0     0     0     0     0      0
  16      2       1.29     0     1     0     0     0     0     0     0     0     0      0
  17      3       0.99     0     0     1     0     0     0     0     0     0     0      0
  18      3       1.04     0     0     1     0     0     0     0     0     0     0      0
  19      3       1.09     0     0     1     0     0     0     0     0     0     0      0
  20      3       1.14     0     0     1     0     0     0     0     0     0     0      0
  21      3       1.19     0     0     1     0     0     0     0     0     0     0      0
  22      3       1.24     0     0     1     0     0     0     0     0     0     0      0
  23      4       0.89     0     0     0     1     0     0     0     0     0     0      0
  24      4       0.94     0     0     0     1     0     0     0     0     0     0      0
  25      4       0.99     0     0     0     1     0     0     0     0     0     0      0
  26      4       1.04     0     0     0     1     0     0     0     0     0     0      0
  27      4       1.09     0     0     0     1     0     0     0     0     0     0      0
  28      4       1.14     0     0     0     1     0     0     0     0     0     0      0
  29      5       1.04     0     0     0     0     1     0     0     0     0     0      0
  30      5       1.09     0     0     0     0     1     0     0     0     0     0      0
  31      5       1.14     0     0     0     0     1     0     0     0     0     0      0
  32      5       1.19     0     0     0     0     1     0     0     0     0     0      0
  33      5       1.24     0     0     0     0     1     0     0     0     0     0      0
  34      5       1.29     0     0     0     0     1     0     0     0     0     0      0
  35      6       0.89     0     0     0     0     0     1     0     0     0     0      0
  36      6       0.99     0     0     0     0     0     1     0     0     0     0      0
  37      6       1.09     0     0     0     0     0     1     0     0     0     0      0
  38      6       1.19     0     0     0     0     0     1     0     0     0     0      0
  39      7       0.99     0     0     0     0     0     0     1     0     0     0      0
  40      7       1.09     0     0     0     0     0     0     1     0     0     0      0
  41      7       1.19     0     0     0     0     0     0     1     0     0     0      0
  42      7       1.29     0     0     0     0     0     0     1     0     0     0      0
  43      8       0.94     0     0     0     0     0     0     0     1     0     0      0
  44      8       0.99     0     0     0     0     0     0     0     1     0     0      0
  45      8       1.14     0     0     0     0     0     0     0     1     0     0      0
  46      8       1.19     0     0     0     0     0     0     0     1     0     0      0
```

```
    47       9     1.09    0    0    0    0    0    0    0    0    1    0     0
    48       9     1.14    0    0    0    0    0    0    0    0    1    0     0
    49       9     1.19    0    0    0    0    0    0    0    0    1    0     0
    50       9     1.24    0    0    0    0    0    0    0    0    1    0     0
    51      10     1.14    0    0    0    0    0    0    0    0    0    1     0
    52      10     1.19    0    0    0    0    0    0    0    0    0    1     0
    53      10     1.24    0    0    0    0    0    0    0    0    0    1     0
    54      10     1.29    0    0    0    0    0    0    0    0    0    1     0
    55    None       .     0    0    0    0    0    0    0    0    0    0     1
```

It begins with eight candidates for the eight prices for the first brand (`Brand` = 1 `f1` = 1, `f2-f11` = 0). It is followed by eight alternatives for the eight prices for the second brand (`Brand` = 2 `f2` = 1, `f1` = 0, `f3` through `f11` = 0). The constant alternative is at the end. The following steps create and display the candidate design:

```
proc format;
   value bf 11 = 'None';
   run;

data cand(keep=brand price f:);
   retain Brand Price f1-f11 0;
   array p[8];
   array f[11];
   infile cards missover;
   input Brand p1-p8;
   do i = 1 to 8;
      Price = p[i];
      if n(price) or (i = 1 and brand = 11) then do;
         f[brand] = 1; output; f[brand] = 0;
         end;
      end;
   format brand bf.;

   datalines;
 1 0.89 0.94 0.99 1.04 1.09 1.14 1.19 1.24
 2 0.94 0.99 1.04 1.09 1.14 1.19 1.24 1.29
 3 0.99 1.04 1.09 1.14 1.19 1.24
 4 0.89 0.94 0.99 1.04 1.09 1.14
 5 1.04 1.09 1.14 1.19 1.24 1.29
 6 0.89 0.99 1.09 1.19
 7 0.99 1.09 1.19 1.29
 8 0.94 0.99 1.14 1.19
 9 1.09 1.14 1.19 1.24
10 1.14 1.19 1.24 1.29
11
;

proc print; run;
```

The PROC FORMAT step creates a format so that the constant alternative, Brand 11, displays as "None". The DATA step creates the candidate alternatives. The `retain` statement names the variables `Brand` and `Price` first so that they appear in the data set first. The important computational reason for the `retain` statement is it retains the values of the variables `f1-f11` across the passes through the DATA step (rather than setting them to missing each time) and it initializes their values to zero. The first array statement creates an array for the eight price variables, `p1-p8` that are read with the `input` statement. The second array statement creates an array for the eleven flag variables, `f1-f11`, that flag which candidates can be used for each of the 11 alternatives. The `infile` statement with the option `missover` is used so that missing values are automatically provided for lines that do not have eight prices. The `do` statement is used to write each price out as a separate observation in the output data set. The *ith* price is stored in the variable `Price`, and it is written to the output data set if it is not missing. In addition, one missing price value is written to the output data set for the constant alternative. The *ith* flag variable is set to 1 before before the *ith* brand is written to the output data set and its value is restored to zero before going on to the next observation.

The following `%ChoicEff` macro step creates the design, naming `Brand` and `Price` as classification variables:

```
%choiceff(data=cand,                /* candidate set of alternatives       */
          model=class(brand         /* model with brand and                */
                      brand*price / /* brand by price effects              */
                      zero=none) /  /* zero=none - use all levels          */
                      lprefix=0     /* use just levels in variable labels  */
                      cprefix=1,    /* use one var name char in new names  */
          nsets=24,                 /* number of choice sets               */
          flags=f1-f11,             /* flag which alt can go where, 11 alts */
          seed=462,                 /* random number seed                  */
          beta=zero)                /* assumed beta vector, Ho: b=0        */
```

Indicator variables are created for all nonmissing levels of the factors. Some of the results are as follows:

```
Design   Iteration  D-Efficiency       D-Error
----------------------------------------------
     1       0                0              .
             1                0              .
                        0.00139 (Ridged)
             2                0              .
                        0.00139 (Ridged)

Design   Iteration  D-Efficiency       D-Error
----------------------------------------------
     2       0                0              .
             1                0              .
                        0.00140 (Ridged)
             2                0              .
                        0.00140 (Ridged)
```

```
                         Final Results

                   Design                  1
                   Choice Sets            24
                   Alternatives           11
                   Parameters             54
                   Maximum Parameters    240
                   D-Efficiency            0
                   D-Error                 .


      Variable                                           Standard
 n    Name          Label          Variance    DF       Error

  1   B1              1              6.5759     1     2.56436
  2   B2              2              4.5072     1     2.12303
  3   B3              3              4.5405     1     2.13086
  4   B4              4              2.8785     1     1.69660
  5   B5              5              3.4751     1     1.86415
  6   B6              6              3.4899     1     1.86812
  7   B7              7              2.8850     1     1.69854
  8   B8              8              2.8778     1     1.69640
  9   B9              9              3.5155     1     1.87497
 10   B10            10              2.8860     1     1.69883
 11   BNone         None               .        0        .
 12   B1P0D89         1 * 0.89       9.2255     1     3.03736
 13   B1P0D94         1 * 0.94      12.2905     1     3.50578
 14   B1P0D99         1 * 0.99      10.2609     1     3.20327
 15   B1P1D04         1 * 1.04      18.6877     1     4.32293
 16   B1P1D09         1 * 1.09      18.6007     1     4.31286
 17   B1P1D14         1 * 1.14       8.2432     1     2.87109
 18   B1P1D19         1 * 1.19       8.6498     1     2.94105
 19   B1P1D24         1 * 1.24         .        0        .
 20   B1P1D29         1 * 1.29         .        0        .
 21   B2P0D89         2 * 0.89         .        0        .
 22   B2P0D94         2 * 0.94       8.1669     1     2.85778
 23   B2P0D99         2 * 0.99       7.1946     1     2.68227
 24   B2P1D04         2 * 1.04       8.2091     1     2.86515
 25   B2P1D09         2 * 1.09       8.2183     1     2.86676
 26   B2P1D14         2 * 1.14      10.3850     1     3.22258
 27   B2P1D19         2 * 1.19       7.1970     1     2.68272
 28   B2P1D24         2 * 1.24      10.2743     1     3.20536
 29   B2P1D29         2 * 1.29         .        0        .
```

```
    30    B3P0D89         3 * 0.89         .         0        .
    31    B3P0D94         3 * 0.94         .         0        .
    32    B3P0D99         3 * 0.99      7.2137       1     2.68584
    33    B3P1D04         3 * 1.04      7.2159       1     2.68624
    34    B3P1D09         3 * 1.09      6.2137       1     2.49273
    35    B3P1D14         3 * 1.14      7.2513       1     2.69283
    36    B3P1D19         3 * 1.19      8.2380       1     2.87020
    37    B3P1D24         3 * 1.24         .         0        .
    38    B3P1D29         3 * 1.29         .         0        .
    39    B4P0D89         4 * 0.89      4.5410       1     2.13097
    40    B4P0D94         4 * 0.94      6.5975       1     2.56855
    41    B4P0D99         4 * 0.99      8.6131       1     2.93480
    42    B4P1D04         4 * 1.04      6.6153       1     2.57202
    43    B4P1D09         4 * 1.09      4.9623       1     2.22762
    44    B4P1D14         4 * 1.14         .         0        .
    45    B4P1D19         4 * 1.19         .         0        .
    46    B4P1D24         4 * 1.24         .         0        .
    47    B4P1D29         4 * 1.29         .         0        .
    48    B5P0D89         5 * 0.89         .         0        .
    49    B5P0D94         5 * 0.94         .         0        .
    50    B5P0D99         5 * 0.99         .         0        .
    51    B5P1D04         5 * 1.04      6.1601       1     2.48195
    52    B5P1D09         5 * 1.09      7.2013       1     2.68352
    53    B5P1D14         5 * 1.14      5.4956       1     2.34427
    54    B5P1D19         5 * 1.19      6.1440       1     2.47872
    55    B5P1D24         5 * 1.24      6.1582       1     2.48157
    56    B5P1D29         5 * 1.29         .         0        .
    57    B6P0D89         6 * 0.89      4.8869       1     2.21063
    58    B6P0D94         6 * 0.94         .         0        .
    59    B6P0D99         6 * 0.99      4.6185       1     2.14906
    60    B6P1D04         6 * 1.04         .         0        .
    61    B6P1D09         6 * 1.09      5.5433       1     2.35442
    62    B6P1D14         6 * 1.14         .         0        .
    63    B6P1D19         6 * 1.19         .         0        .
    64    B6P1D24         6 * 1.24         .         0        .
    65    B6P1D29         6 * 1.29         .         0        .
    66    B7P0D89         7 * 0.89         .         0        .
    67    B7P0D94         7 * 0.94         .         0        .
    68    B7P0D99         7 * 0.99      4.2574       1     2.06333
    69    B7P1D04         7 * 1.04         .         0        .
    70    B7P1D09         7 * 1.09      4.9384       1     2.22225
    71    B7P1D14         7 * 1.14         .         0        .
    72    B7P1D19         7 * 1.19      4.2492       1     2.06136
    73    B7P1D24         7 * 1.24         .         0        .
    74    B7P1D29         7 * 1.29         .         0        .
```

```
      75    B8P0D89          8 * 0.89          .         0        .
      76    B8P0D94          8 * 0.94     4.2680         1     2.06592
      77    B8P0D99          8 * 0.99     4.2502         1     2.06161
      78    B8P1D04          8 * 1.04          .         0        .
      79    B8P1D09          8 * 1.09          .         0        .
      80    B8P1D14          8 * 1.14     4.9127         1     2.21645
      81    B8P1D19          8 * 1.19          .         0        .
      82    B8P1D24          8 * 1.24          .         0        .
      83    B8P1D29          8 * 1.29          .         0        .
      84    B9P0D89          9 * 0.89          .         0        .
      85    B9P0D94          9 * 0.94          .         0        .
      86    B9P0D99          9 * 0.99          .         0        .
      87    B9P1D04          9 * 1.04          .         0        .
      88    B9P1D09          9 * 1.09     5.5866         1     2.36360
      89    B9P1D14          9 * 1.14     4.6559         1     2.15775
      90    B9P1D19          9 * 1.19     4.9117         1     2.21623
      91    B9P1D24          9 * 1.24          .         0        .
      92    B9P1D29          9 * 1.29          .         0        .
      93    B10P0D89        10 * 0.89          .         0        .
      94    B10P0D94        10 * 0.94          .         0        .
      95    B10P0D99        10 * 0.99          .         0        .
      96    B10P1D04        10 * 1.04          .         0        .
      97    B10P1D09        10 * 1.09          .         0        .
      98    B10P1D14        10 * 1.14     4.9756         1     2.23060
      99    B10P1D19        10 * 1.19     4.0131         1     2.00327
     100    B10P1D24        10 * 1.24     4.5461         1     2.13216
     101    B10P1D29        10 * 1.29          .         0        .
     102    BNoneP0D89    None * 0.89          .         0        .
     103    BNoneP0D94    None * 0.94          .         0        .
     104    BNoneP0D99    None * 0.99          .         0        .
     105    BNoneP1D04    None * 1.04          .         0        .
     106    BNoneP1D09    None * 1.09          .         0        .
     107    BNoneP1D14    None * 1.14          .         0        .
     108    BNoneP1D19    None * 1.19          .         0        .
     109    BNoneP1D24    None * 1.24          .         0        .
     110    BNoneP1D29    None * 1.29          .         0        .
                                                        ==
                                                        54
```

There are unneeded parameters in our model, and for the moment, that is fine. We see 10 parameters for `Brand`. The constant alternative is the reference alternative. We see 7 parameters for Brand 1's price (8 prices minus 1 = 7), 7 parameters for Brand 2's price, 5 parameters for Brand 3's price (6 prices minus 1 = 5), ..., and 3 parameters for Brand 10's price (4 prices minus 1 = 3). This all looks correct.

The following redundant variable list is displayed in the log:

```
Redundant Variables:

zBNone B1P1D24 B1P1D29 B2P0D89 B2P1D29 B3P0D89 B3P0D94 B3P1D24 B3P1D29 B4P1D14
B4P1D19 B4P1D24 B4P1D29 B5P0D89 B5P0D94 B5P0D99 B5P1D29 B6P0D94 B6P1D04 B6P1D14
B6P1D19 B6P1D24 B6P1D29 B7P0D89 B7P0D94 B7P1D04 B7P1D14 B7P1D24 B7P1D29 B8P0D89
B8P1D04 B8P1D09 B8P1D19 B8P1D24 B8P1D29 B9P0D89 B9P0D94 B9P0D99 B9P1D04 B9P1D24
B9P1D29 B10P0D89 B10P0D94 B10P0D99 B10P1D04 B10P1D09 B10P1D29 BNoneP0D89
BNoneP0D94 BNoneP0D99 BNoneP1D04 BNoneP1D09 BNoneP1D14 BNoneP1D19 BNoneP1D24
BNoneP1D29
```

The following list is the same, except it has been manually reformatted to group the brands:

```
Redundant Variables:

B1P1D24 B1P1D29
B2P0D89 B2P1D29
B3P0D89 B3P0D94 B3P1D24 B3P1D29
B4P1D14 B4P1D19 B4P1D24 B4P1D29
B5P0D89 B5P0D94 B5P0D99 B5P1D29
B6P0D94 B6P1D04 B6P1D14 B6P1D19 B6P1D24 B6P1D29
B7P0D89 B7P0D94 B7P1D04 B7P1D14 B7P1D24 B7P1D29
B8P0D89 B8P1D04 B8P1D09 B8P1D19 B8P1D24 B8P1D29
B9P0D89 B9P0D94 B9P0D99 B9P1D04 B9P1D24 B9P1D29
B10P0D89 B10P0D94 B10P0D99 B10P1D04 B10P1D09 B10P1D29
BNoneP0D89 BNoneP0D94 BNoneP0D99 BNoneP1D04 BNoneP1D09 BNoneP1D14
BNoneP1D19 BNoneP1D24 BNoneP1D29
```

For Brands 1 and 2, we have 7 parameters and the last level for price (8) is the reference level and does not appear in the model. The specification `class(brand brand*price / zero=none)` suppresses having a reference level for brand so that all 10 brands appear along with the constant alternative.

For all brands, the reference level is the last price in the list: 1.24 for brands 1, 3 and 9; 1.29 for brands 2, 5, 7, and 10; and so on. In addition, missing variances and 0 *df* are displayed for each price that does not appear with a brand. Every parameter associated with the constant alternative has a missing variance and 0 *df*. The following step creates the design:

```
%let vars=
BNone B1P1D24 B1P1D29 B2P0D89 B2P1D29 B3P0D89 B3P0D94 B3P1D24 B3P1D29 B4P1D14
B4P1D19 B4P1D24 B4P1D29 B5P0D89 B5P0D94 B5P0D99 B5P1D29 B6P0D94 B6P1D04 B6P1D14
B6P1D19 B6P1D24 B6P1D29 B7P0D89 B7P0D94 B7P1D04 B7P1D14 B7P1D24 B7P1D29 B8P0D89
B8P1D04 B8P1D09 B8P1D19 B8P1D24 B8P1D29 B9P0D89 B9P0D94 B9P0D99 B9P1D04 B9P1D24
B9P1D29 B10P0D89 B10P0D94 B10P0D99 B10P1D04 B10P1D09 B10P1D29 BNoneP0D89
BNoneP0D94 BNoneP0D99 BNoneP1D04 BNoneP1D09 BNoneP1D14 BNoneP1D19 BNoneP1D24
BNoneP1D29;
```

```
%choiceff(data=cand,                /* candidate set of alternatives       */
          model=class(brand         /* model with brand and                */
                      brand*price / /* brand by price effects              */
                      zero=none) /  /* zero=none - use all levels          */
                      lprefix=0     /* use just levels in variable labels  */
                      cprefix=1,    /* use one var name char in new names  */
          drop=&vars,               /* extra terms to drop                 */
          nsets=24,                 /* number of choice sets               */
          flags=f1-f11,             /* flag which alt can go where, 11 alts */
          seed=462,                 /* random number seed                  */
          beta=zero)                /* assumed beta vector, Ho: b=0        */
```

Some of the results are as follows:

```
Design   Iteration  D-Efficiency      D-Error
----------------------------------------------
     1       0          0.31384 *     3.18632
             1          0.34074 *     2.93476
             2          0.34074       2.93476

Design   Iteration  D-Efficiency      D-Error
----------------------------------------------
     2       0                0             .
             1          0.34101 *     2.93246

                Final Results

          Design                  2
          Choice Sets            24
          Alternatives           11
          Parameters             54
          Maximum Parameters    240
          D-Efficiency       0.3410
          D-Error            2.9325
```

```
                                                          Standard
  n    Variable Name    Label                Variance    DF     Error

  1    Brand1           Brand  1              4.50417     1    2.12230
  2    Brand2           Brand  2              4.52567     1    2.12736
  3    Brand3           Brand  3              3.47776     1    1.86487
  4    Brand4           Brand  4              3.48724     1    1.86742
  5    Brand5           Brand  5              3.49982     1    1.87078
  6    Brand6           Brand  6              2.47337     1    1.57270
  7    Brand7           Brand  7              2.45738     1    1.56760
  8    Brand8           Brand  8              2.45142     1    1.56570
  9    Brand9           Brand  9              2.45282     1    1.56615
 10    Brand10          Brand 10              2.44575     1    1.56389
 11    Brand1Price1     Brand  1 * Price 1    8.19264     1    2.86228
 12    Brand1Price2     Brand  1 * Price 2    8.19269     1    2.86229
 13    Brand1Price3     Brand  1 * Price 3    8.18940     1    2.86171
 14    Brand1Price4     Brand  1 * Price 4    8.23067     1    2.86891
 15    Brand1Price5     Brand  1 * Price 5    8.21587     1    2.86633
 16    Brand1Price6     Brand  1 * Price 6    8.19365     1    2.86246
 17    Brand1Price7     Brand  1 * Price 7    8.23031     1    2.86885
 18    Brand2Price1     Brand  2 * Price 1    8.16830     1    2.85802
 19    Brand2Price2     Brand  2 * Price 2    8.23185     1    2.86912
 20    Brand2Price3     Brand  2 * Price 3    8.21687     1    2.86651
 21    Brand2Price4     Brand  2 * Price 4    8.23295     1    2.86931
 22    Brand2Price5     Brand  2 * Price 5    8.27059     1    2.87586
 23    Brand2Price6     Brand  2 * Price 6    8.22612     1    2.86812
 24    Brand2Price7     Brand  2 * Price 7    8.28203     1    2.87785
 25    Brand3Price1     Brand  3 * Price 1    6.15558     1    2.48104
 26    Brand3Price2     Brand  3 * Price 2    6.17640     1    2.48524
 27    Brand3Price3     Brand  3 * Price 3    6.13255     1    2.47640
 28    Brand3Price4     Brand  3 * Price 4    6.14840     1    2.47960
 29    Brand3Price5     Brand  3 * Price 5    6.11249     1    2.47234
 30    Brand4Price1     Brand  4 * Price 1    6.17231     1    2.48441
 31    Brand4Price2     Brand  4 * Price 2    6.22760     1    2.49552
 32    Brand4Price3     Brand  4 * Price 3    6.12111     1    2.47409
 33    Brand4Price4     Brand  4 * Price 4    6.19792     1    2.48956
 34    Brand4Price5     Brand  4 * Price 5    6.12131     1    2.47413
 35    Brand5Price1     Brand  5 * Price 1    6.21514     1    2.49302
 36    Brand5Price2     Brand  5 * Price 2    6.15748     1    2.48143
 37    Brand5Price3     Brand  5 * Price 3    6.17697     1    2.48535
 38    Brand5Price4     Brand  5 * Price 4    6.16121     1    2.48218
 39    Brand5Price5     Brand  5 * Price 5    6.20067     1    2.49011
 40    Brand6Price1     Brand  6 * Price 1    4.16170     1    2.04002
 41    Brand6Price2     Brand  6 * Price 2    4.11324     1    2.02811
 42    Brand6Price3     Brand  6 * Price 3    4.13298     1    2.03297
 43    Brand7Price1     Brand  7 * Price 1    4.10703     1    2.02658
 44    Brand7Price2     Brand  7 * Price 2    4.11083     1    2.02752
 45    Brand7Price3     Brand  7 * Price 3    4.10632     1    2.02641
```

```
      46    Brand8Price1     Brand  8 * Price 1     4.12107     1     2.03004
      47    Brand8Price2     Brand  8 * Price 2     4.10075     1     2.02503
      48    Brand8Price3     Brand  8 * Price 3     4.08366     1     2.02081
      49    Brand9Price1     Brand  9 * Price 1     4.11157     1     2.02770
      50    Brand9Price2     Brand  9 * Price 2     4.10049     1     2.02497
      51    Brand9Price3     Brand  9 * Price 3     4.10522     1     2.02614
      52    Brand10Price1    Brand 10 * Price 1     4.07896     1     2.01964
      53    Brand10Price2    Brand 10 * Price 2     4.09065     1     2.02253
      54    Brand10Price3    Brand 10 * Price 3     4.11148     1     2.02768
                                                               ==
                                                               54
```

We can see that we now have all the terms for the final model. The following step displays part of the design:

```
proc print data=best(obs=22); id set; by set; var brand price; run;
```

The first two choice sets are as follows:

```
                     Set    Brand    Price

                      1        1      0.94
                               2      1.29
                               3      1.24
                               4      1.14
                               5      1.19
                               6      1.19
                               7      1.09
                               8      1.19
                               9      1.24
                              10      1.24
                            None        .

                      2        1      1.04
                               2      1.29
                               3      1.14
                               4      0.99
                               5      1.09
                               6      1.19
                               7      1.29
                               8      1.14
                               9      1.24
                              10      1.24
                            None        .
```

Alternative Swapping with Price Constraints and Discounts

This example finds a choice design where there are constraints on the price factor. Specifically, for the first alternative, price can have one of three values, $0.75, $1.00, or $1.25. In the second alternative, price will either be a 3%, 4%, or 5% discount of the alternative one price. In the third alternative, price will either be a 6%, 7%, or 8% discount of the alternative one price. This example also has 6 two-level factors, which are used for other attributes. These other attributes require no special handling. Since the goal is to produce a price attribute with price dependencies across alternatives, we will construct a candidate set of choice sets and build the design from that. First, we use the `%MktRuns` macro to get suggestions about candidate set sizes as follows:

```
%mktruns(3 2 ** 6  3 2 ** 6  3 2 ** 6)
```

The levels specification has a "3" for the price attribute for the first alternative, a "3" for the 3 discounts for the second alternative, and a "3" for the 3 discounts for the third alternative. The two-level factors make up the other 6 attributes for each of the three alternatives.

Some of the results are as follows:

```
                              Design Summary

                          Number of
                          Levels       Frequency

                               2          18
                               3           3

                 Saturated      = 25
                 Full Factorial = 7,077,888

                 Some Reasonable                      Cannot Be
                    Design Sizes       Violations     Divided By

                              36                0
                              72 *              0
                              48                3     9
                              60                3     9
                              28               60     3  6  9
                              32               60     3  6  9
                              40               60     3  6  9
                              44               60     3  6  9
                              52               60     3  6  9
                              56               60     3  6  9
                              25 S            231     2  3  4  6  9
```

```
            * - 100% Efficient design can be made with the MktEx macro.
            S - Saturated Design - The smallest design that can be made.
                Note that the saturated design is not one of the
                recommended designs for this problem.  It is shown
                to provide some context for the recommended sizes.
```

The results show that an orthogonal array exists in 72 runs, so we will try that first using the %MktEx macro as follows:

```
%mktex(3 2 ** 6  3 2 ** 6  3 2 ** 6, n=72, seed=289)
```

Some of the results are as follows:

```
                              Algorithm Search History

                                     Current          Best
             Design    Row,Col  D-Efficiency  D-Efficiency  Notes
             ----------------------------------------------------------
                  1      Start      100.0000      100.0000  Tab
```

The %MktEx macro found a 100% efficient orthogonal array. The "Tab" note in the algorithm search history in a line that displays 100% efficiency shows that the design was directly constructed from the %MktEx macro's table or catalog of orthogonal designs. Next, the price factors (x1, x8, and x15) with levels 1, 2, and 3 need to be converted to actual prices. In addition, the discounts are stored in three new factors (d1, d2, and d3. We do not really need these factors; they will just be created for our reference. The following step does the conversion and displays the first 10 choice sets:

```
data cand;                           /* new candidates with recoded price    */
   set randomized;                   /* use randomized design from MktEx     */
   x1  = 0.5 + 0.25 * x1;            /* map 1, 2, 3 to 0.75, 1.0, 1.25       */
   d1  = 0;                          /* discount for first alt is no discount*/
   d2  = 2 + x8;                     /* map 1, 2, 3 to 3%, 4%, 5% discount   */
   d3  = 5 + x15;                    /* map 1, 2, 3 to 6%, 7%, 8% discount   */
   x8  = round(x1 * (1 - d2 / 100), 0.01);/* apply discount to second alt   */
   x15 = round(x1 * (1 - d3 / 100), 0.01);/* apply discount to third alt    */
   run;

proc print data=cand(obs=10); run;
```

The first 10 candidates are as follows:

```
 O                        x x x x x   x  x x x x x x
 b   x  x x x x x x   x  x 1 1 1 1 1   1  1 1 1 1 2 2 d d d
 s   1  2 3 4 5 6 7   8  9 0 1 2 3 4   5  6 7 8 9 0 1 1 2 3

 1 0.75 1 2 2 1 1 1 0.72 2 2 2 2 1 2 0.70 2 2 1 2 1 1 0 4 7
 2 1.00 1 1 2 1 2 1 0.96 2 2 2 1 2 1 0.94 2 1 1 2 2 2 0 4 6
 3 1.00 1 1 1 1 1 1 0.97 2 2 2 1 1 2 0.93 1 1 2 1 1 1 0 3 7
 4 1.00 2 2 2 1 2 1 0.96 2 1 1 2 2 1 0.94 2 1 2 1 1 1 0 4 6
 5 0.75 2 1 2 2 2 1 0.72 2 1 1 1 1 2 0.70 2 2 2 2 2 1 0 4 7
 6 1.25 1 2 2 1 2 2 1.21 1 1 1 2 1 2 1.18 1 2 2 1 2 1 0 3 6
 7 1.00 1 1 2 2 2 2 0.97 2 2 1 1 2 2 0.93 2 1 2 2 1 2 0 3 7
 8 1.25 2 2 2 1 2 1 1.20 2 2 1 2 1 2 1.15 1 1 1 2 2 2 0 4 8
 9 1.25 2 1 1 1 1 2 1.20 2 1 1 2 2 1 1.15 1 1 2 2 2 2 0 4 8
10 1.25 2 2 2 2 1 2 1.19 1 1 2 1 2 2 1.16 1 2 1 2 1 2 0 5 7
```

Next, our goal is to convert our linear candidate design into a choice candidate design. We will need a `Key` data set that provides the rules for conversion, and the `%MktKey` macro can help us by constructing part of this data set. The following step creates the names `x1-x21` in a 3×7 array as follows:

```
%mktkey(3 7)
```

The results are as follows:

```
x1      x2      x3      x4      x5      x6      x7

x1      x2      x3      x4      x5      x6      x7
x8      x9      x10     x11     x12     x13     x14
x15     x16     x17     x18     x19     x20     x21
```

We will use these results to construct and display the `Key` data set as follows:

```
data key;
   input (Price x2-x7 Discount) ($);
   datalines;
x1     x2     x3     x4     x5     x6     x7   d1
x8     x9     x10    x11    x12    x13    x14  d2
x15    x16    x17    x18    x19    x20    x21  d3
;

proc print; run;
```

The results are as follows:

```
         Obs    Price    x2     x3     x4     x5     x6     x7     Discount

          1      x1      x2     x3     x4     x5     x6     x7        d1
          2      x8      x9     x10    x11    x12    x13    x14       d2
          3      x15     x16    x17    x18    x19    x20    x21       d3
```

The price attribute is made from `x1`, `x8`, and `x15`. The discount (informational only) attribute is made from `d1`, `d2`, and `d3`). The other attributes in the choice design are made from the other factors in the linear arrangement in the usual way. We use this information to create the choice design from our linear arrangement that has the actual prices and the discounts as follows:

```
%mktroll(design=cand, key=key, out=cand2)

proc print data=cand2(obs=30); id set; by set; run;
```

The first 10 candidate choice sets are as follows:

```
       Set    _Alt_    Price    x2    x3    x4    x5    x6    x7    Discount

        1       1       0.75     1     2     2     1     1     1        0
                2       0.72     2     2     2     2     1     2        4
                3       0.70     2     2     1     2     1     1        7

        2       1       1.00     1     1     2     1     2     1        0
                2       0.96     2     2     2     1     2     1        4
                3       0.94     2     1     1     2     2     2        6

        3       1       1.00     1     1     1     1     1     1        0
                2       0.97     2     2     2     1     1     2        3
                3       0.93     1     1     2     1     1     1        7

        4       1       1.00     2     2     2     1     2     1        0
                2       0.96     2     1     1     2     2     1        4
                3       0.94     2     1     2     1     1     1        6

        5       1       0.75     2     1     2     2     2     1        0
                2       0.72     2     1     1     1     1     2        4
                3       0.70     2     2     2     2     2     1        7

        6       1       1.25     1     2     2     1     2     2        0
                2       1.21     1     1     1     2     1     2        3
                3       1.18     1     2     2     1     2     1        6

        7       1       1.00     1     1     2     2     2     2        0
                2       0.97     2     2     1     1     2     2        3
                3       0.93     2     1     2     2     1     2        7
       8       1       1.25     2     2     2     1     2     1        0
               2       1.20     2     2     1     2     1     2        4
               3       1.15     1     1     1     2     2     2        8

       9       1       1.25     2     1     1     1     1     2        0
               2       1.20     2     1     1     2     2     1        4
               3       1.15     1     1     2     2     2     2        8

      10       1       1.25     2     2     2     2     1     2        0
               2       1.19     1     1     2     1     2     2        5
               3       1.16     1     2     1     2     1     2        7
```

Next, we use the `%ChoicEff` macro to search for a design as follows:

```
%choiceff(data=cand2,                /* candidate set of choice sets         */
          model=ide(Price)           /* model with quantitative price effect */
                class(x2-x7 / effects), /* binary attributes, effects coded */
          nsets=20,                  /* 20 choice sets                       */
          nalts=3,                   /* 3 alternatives per set               */
          morevars=discount,         /* add this var to the output data set  */
          drop=discount,             /* do not add this var to the model     */
          seed=292,                  /* random number seed                   */
          options=nodups,            /* no duplicate choice sets             */
          maxiter=10,                /* maximum number of designs to create  */
          beta=zero)                 /* assumed beta vector, Ho: b=0         */
```

A subset of the results are as follows:

```
                              Final Results

                        Design                  1
                        Choice Sets            20
                        Alternatives            3
                        Parameters              7
                        Maximum Parameters     40
                        D-Efficiency       6.2005
                        D-Error            0.1613
```

```
              Variable                                      Standard
         n      Name      Label    Variance     DF       Error

         1     Price      Price     41.7548      1     6.46179
         2     x21        x2 1       0.0674      1     0.25953
         3     x31        x3 1       0.0687      1     0.26204
         4     x41        x4 1       0.0600      1     0.24489
         5     x51        x5 1       0.0664      1     0.25762
         6     x61        x6 1       0.0570      1     0.23866
         7     x71        x7 1       0.0693      1     0.26329
                                                ==
                                                 7
```

You can display the covariances as follows:

```
proc print data=bestcov label;
   title 'Variance-Covariance Matrix';
   id __label;
   label __label = '00'x;
   var price x:;
   run;
title;
```

The results are as follows:

```
                              Variance-Covariance Matrix

                Price    x2 1      x3 1      x4 1      x5 1      x6 1      x7 1

      Price   41.7548  0.055064 -0.045773  0.016390 -0.043134  0.001150  0.076607
      x2 1     0.0551  0.067356 -0.002937 -0.003431  0.001635  0.003638  0.006400
      x3 1    -0.0458 -0.002937  0.068664 -0.000748  0.000304  0.003032 -0.012407
      x4 1     0.0164 -0.003431 -0.000748  0.059972  0.001552 -0.004627  0.003199
      x5 1    -0.0431  0.001635  0.000304  0.001552  0.066369 -0.000065 -0.001835
      x6 1     0.0012  0.003638  0.003032 -0.004627 -0.000065  0.056960  0.001193
      x7 1     0.0766  0.006400 -0.012407  0.003199 -0.001835  0.001193  0.069321
```

Clearly, the variance for the price attribute is much greater than the variances for the other attributes. This is not surprising. Two points define a line. Having more points is inefficient. We can see how many price points we actually have as follows:

```
proc freq data=best; tables price; run;
```

The results are as follows:

```
                              The FREQ Procedure

                                                Cumulative    Cumulative
                Price    Frequency     Percent     Frequency      Percent
                ----------------------------------------------------------
                 0.69           1        1.67             1         1.67
                  0.7           1        1.67             2         3.33
                 0.71           4        6.67             6        10.00
                 0.72           1        1.67             7        11.67
                 0.73           1        1.67             8        13.33
                 0.75           4        6.67            12        20.00
                 0.92           3        5.00            15        25.00
                 0.94           1        1.67            16        26.67
                 0.95           3        5.00            19        31.67
                 0.96           1        1.67            20        33.33
                    1           4        6.67            24        40.00
                 1.15           6       10.00            30        50.00
                 1.16           6       10.00            36        60.00
                 1.19           6       10.00            42        70.00
                  1.2           6       10.00            48        80.00
                 1.25          12       20.00            60       100.00
```

Note that the `%ChoicEff` macro selects many more $1.25's than the other levels. Two extreme points define a line. It has a clear maximum it can grab and try to "load up" on. The minimum is a bit fuzzier. Alternatively, you can treat the price attribute as a classification variable as follows:

```
   %choiceff(data=cand2,                /* candidate set of choice sets         */
             model=class(Price / zero=none)/* model with qualitative price     */
                   class(x2-x7 / effects), /* binary attributes, effects coded */
             nsets=20,                  /* 20 choice sets                       */
             nalts=3,                   /* 3 alternatives per set               */
             morevars=discount,         /* add this var to the output data set  */
             drop=discount,             /* do not add this var to the model     */
             seed=292,                  /* random number seed                   */
             options=nodups,            /* no duplicate choice sets             */
             maxiter=10,                /* maximum number of designs to create  */
             beta=zero)                 /* assumed beta vector, Ho: b=0         */
```

Some of the results are as follows:

```
                          Final Results

                    Design                   1
                    Choice Sets             20
                    Alternatives             3
                    Parameters              23
                    Maximum Parameters      40
                    D-Efficiency             0
                    D-Error                  .

          Variable                                      Standard
     n    Name         Label          Variance    DF       Error

     1    Price0D69    Price 0.69      3.00000     1     1.73205
     2    Price0D7     Price  0.7      2.18594     1     1.47849
     3    Price0D71    Price 0.71      2.25000     1     1.50000
     4    Price0D72    Price 0.72      2.00000     1     1.41421
     5    Price0D73    Price 0.73      3.00000     1     1.73205
     6    Price0D75    Price 0.75       .          0      .
     7    Price0D92    Price 0.92      3.00000     1     1.73205
     8    Price0D93    Price 0.93      2.06302     1     1.43632
     9    Price0D94    Price 0.94      3.00000     1     1.73205
    10    Price0D95    Price 0.95      3.00000     1     1.73205
    11    Price0D96    Price 0.96      3.00000     1     1.73205
    12    Price0D97    Price 0.97      2.17145     1     1.47358
    13    Price1       Price    1       .          0      .
    14    Price1D15    Price 1.15      3.00000     1     1.73205
    15    Price1D16    Price 1.16      3.00000     1     1.73205
    16    Price1D18    Price 1.18      3.00000     1     1.73205
    17    Price1D19    Price 1.19      3.00000     1     1.73205
    18    Price1D2     Price  1.2      3.00000     1     1.73205
    19    Price1D21    Price 1.21      3.00000     1     1.73205
    20    Price1D25    Price 1.25       .          0      .
    21    x21          x2 1            0.06091     1     0.24681
    22    x31          x3 1            0.08720     1     0.29530
    23    x41          x4 1            0.08171     1     0.28585
    24    x51          x5 1            0.07465     1     0.27322
    25    x61          x6 1            0.07592     1     0.27554
    26    x71          x7 1            0.08752     1     0.29584
                                                  ==
                                                  23
```

It is interesting to note that the price parameters for our alternative 1 levels are all missing with zero *df*. Because the prices for alternatives two and three are discounts of the alternative one price, they come before the alternative one price in the list of sorted prices. Then because of the dependencies in the prices, only the discounted prices are estimable in the model. You can see the price frequencies as follows:

```
proc freq data=best; tables price; run;
```

The results are as follows:

```
                          The FREQ Procedure

                                              Cumulative    Cumulative
             Price    Frequency     Percent     Frequency      Percent
             ---------------------------------------------------------
              0.69           2        3.33             2         3.33
               0.7           3        5.00             5         8.33
              0.71           4        6.67             9        15.00
              0.72           3        5.00            12        20.00
              0.73           2        3.33            14        23.33
              0.75           7       11.67            21        35.00
              0.92           2        3.33            23        38.33
              0.93           3        5.00            26        43.33
              0.94           2        3.33            28        46.67
              0.95           2        3.33            30        50.00
              0.96           2        3.33            32        53.33
              0.97           3        5.00            35        58.33
                 1           7       11.67            42        70.00
              1.15           2        3.33            44        73.33
              1.16           2        3.33            46        76.67
              1.18           2        3.33            48        80.00
              1.19           2        3.33            50        83.33
               1.2           2        3.33            52        86.67
              1.21           2        3.33            54        90.00
              1.25           6       10.00            60       100.00
```

If your goal is a more even price distribution, designating price as a classification variable will work better than designating it as an identity variable. Note that you can design the experiment designating price as a classification variable and analyze it with price as an identity variable. The opposite approach is not guaranteed to work. If you generate a design using a model with one *df* for price, you should not attempt to then fit a model with multiple *df* for price.

You might be able to do better in this example by using a larger candidate set. Multiplying 72 by numbers like 2, 3, 4, or 6 might help, for example, as follows:

```
%mktex(3 2 ** 6  3 2 ** 6  3 2 ** 6, n=2 * 72, seed=289)

%mkteval;
```

If you do this, be sure to look at the *n*-way frequencies in the `%MktEval` output to ensure that you are not simply getting duplicate candidate choice sets. Since the point of this example is to illustrate how to construct the price attribute with discounts and dependencies, and that has already been accomplished, we will not explore varying candidate sets sizes here.

Multiple Alternative and Choice Set Types

This next example is an order of magnitude more complicated than the kinds of design problems that most analysts ever encounter. If you are just learning the `%ChoicEff` macro, you should skip ahead to page 916 and come back to this section later. If instead, you are a sophisticated analyst, this example shows you the power that you have to make complicated designs using the `%ChoicEff` macro and SAS programming statements.

The goal in this example is to create a design for a choice study. Choice sets consist of two different types of alternatives, and the choice study needs to consist of choice sets with an even mix of 6, 7, and 8 alternatives. Each choice set must contain a subset (of size 6, 7, or 8) of the 10 available brands, and a brand may not appear more than once in any given choice set. There are two types of alternatives, because there are two types of brands. Within each type of brand, the numbers of factor are the same, although they could be different. To make all of this clearer, three possible choice sets are as follows:

```
                            Set     Alt
     Set     Alt    Brand   Type    Type    x1    x2    x3    x4    x5    x6

      11       1       1      1       1      2     3     2     2     1     1
               2       2      1       1      1     2     2     1     1     3
               3       3      1       2      2     2     1     2     1     4
               4       5      1       2      5     3     1     3     2     2
               5       7      1       2      1     1     3     3     2     1
               6       9      1       2      4     4     4     1     2     2

      23       1       1      2       1      2     1     2     3     1     4
               2       2      2       1      1     3     3     1     1     1
               3       3      2       2      2     2     2     1     2     1
               4       4      2       2      3     2     1     2     1     1
               5       6      2       2      1     2     4     2     2     3
               6       7      2       2      3     1     3     2     2     4
               7      10      2       2      4     1     2     3     1     2

      56       1       1      3       1      2     2     3     1     2     4
               2       2      3       1      2     3     3     3     2     3
               3       3      3       2      5     4     2     1     1     1
               4       4      3       2      2     1     1     2     2     2
               5       5      3       2      4     1     2     3     1     2
               6       6      3       2      5     2     2     2     2     2
               7       7      3       2      3     3     4     1     1     2
               8       8      3       2      4     4     2     3     1     1
```

The first choice set (set 11) is of type 1 (6 alternatives), the second (set 23) is of type 2 (7 alternatives), and the third (set 56) is of type 3 (8 alternatives). In this design, `x1` will become the price factor, and `x2` will become the product size factor. For Brand 1 and Brand 2 alternatives, `x1` has 2 levels and `x2` has 3 levels. For Brand 3 through Brand 10 alternatives, `x1` has 5 levels and `x2` has 4 levels. For all brands, `x3` has 4 levels, `x4` has 3 levels, `x5` has 2 levels, and `x6` has 4 levels.

With a complicated problem such as this, there is always more than one way to proceed. In this example, we take the following approach:

- Create a candidate set of alternatives for the first type of alternative (Brand 1 and Brand 2).
- Create a candidate set of alternatives for the second type of alternative (Brand 3 through Brand 10).
- Combine the candidate sets and create the brand factor.
- Use the `%ChoicEff` macro to search the candidate set of alternatives and create a set of candidate choice sets with 6 alternatives.
- Use the `%ChoicEff` macro to search the candidate set of alternatives and create a set of candidate choice sets with 7 alternatives.
- Use the `%ChoicEff` macro to search the candidate set of alternatives and create a set of candidate choice sets with 8 alternatives.
- Combine the three individual candidate sets of choice sets into one big candidate set.
- Extract just the choice sets where no brand appears more than once.
- Array the candidate choice sets with a fixed number of alternatives (8). Add all missing alternatives to the choice sets with 6 or 7 alternatives, and provide a weight variable that identifies those alternatives that are actually used (weight = 1) and those alternatives that are not used (weight = 0).
- Search this candidate set of choice sets using the `%ChoicEff` macro.
- Display the final design.
- Check the final design for duplicate choice sets.

The rest of this section works through each of these steps and discusses the programming involved in creating this design.

Candidate Set of Alternatives

In this example, there are two different types of brands. One type is based on current brands and the other type is based on some new proposed brands. The number of factor levels is different for the two types. In this first set of steps, two candidate sets of candidate alternatives are created. All attributes but brand are created now; brand is added in later. The following statements create two sets of candidate alternatives, one with one type of alternative, and the other with the other type of alternative:

```
* Eliminate or replace options=quick when you know what you are doing.
*
* Increase n=.  Small values are good for testing.
* Larger values should give better designs.
;

%mktruns(2 3 4 3 2 4, interact=1*2)

%mktex(2 3 4 3 2 4,                 /* attrs for proposed brands           */
       interact=1*2,                /* x1*x2 interaction                   */
       n=24,                        /* number of candidate alternatives    */
       seed=292,                    /* random number seed                  */
       out=d1,                      /* output experimental design          */
       options=quick)               /* provides a quick run initially      */

%mktruns(5 4 4 3 2 4, interact=1*2)

%mktex(5 4 4 3 2 4,                 /* attrs for current brands            */
       interact=1*2,                /* x1*x2 interaction                   */
       n=40,                        /* number of candidate alternatives    */
       seed=292,                    /* random number seed                  */
       out=d2,                      /* output experimental design          */
       options=quick)               /* provides a quick run initially      */
```

For each of the two types of factors, the `%MktRuns` macro is used to suggest a size for the candidate design, and then the `%MktEx` macro is used to create the candidate alternatives. In both cases, one of the smaller suggestions from the `%MktRuns` macro is used. When you have your code thoroughly tested, then you should consider both specifying larger values for `n=` and removing `options=quick`. Both are specified for now to make these and subsequent steps run faster.

Some of the output from the first `%MktRuns` step is as follows:

```
            Some Reasonable                 Cannot Be
               Design Sizes      Violations     Divided By

                        144               0
                         72               1     16
                         48               2      9 18
                         96               2      9 18
                        192               2      9 18
                         24               3      9 16 18
                        120               3      9 16 18
                        168               3      9 16 18
                         36               7      8 16 24
                        108               7      8 16 24
                         15 S            23      2  4  6  8  9 12 16 18 24
```

Some of the output from the second `%MktRuns` step is as follows:

```
      Some Reasonable                  Cannot Be
         Design Sizes     Violations   Divided By

                  120              5   16 80
                   80              7    3  6 12 15 60
                  160              7    3  6 12 15 60
                   60              9    8 16 40 80
                  180              9    8 16 40 80
                   48             10    5 10 15 20 40 60 80
                   96             10    5 10 15 20 40 60 80
                  144             10    5 10 15 20 40 60 80
                  192             10    5 10 15 20 40 60 80
                   40             12    3  6 12 15 16 60 80
                   29 S           25    2  3  4  5  6  8 10 12 15 16 20 40 60 80
```

Based on these results, candidate sets of alternatives of size 24 and 40 are selected. Later, once the code is tested and working, you should try larger sizes that will make the program run more slowly. They might also result in better designs. These results suggest sizes like 144 and 120.

The `%MktEx` macro is run to make the two candidate designs. In both cases, one two-way interaction, price by product size, is required to be estimable. The results are stored in two data sets, `d1` and `d2`. The designs and iteration histories for these steps are not shown here.

Combine the Candidate Sets

The following step combines the two sets of candidate alternatives into one set:

```
* Create full candidate design.;
data all;
   retain f1-f8 Brand 1;
   set d1(in=d1) d2(in=d2);

   * Make alternative-specific changes to the price
   * and other attribute levels in here as necessary.
   ;

   * For Brand 1 and Brand 2, write out the other attrs;
   if d1 then do;
      do brand = 1 to 2; output; end;
      end;

   * For Brand 3 through Brand 10, write out the other attrs;
   if d2 then do;
      do brand = 3 to 10; output; end;
      end;
   run;
```

Subsequent steps use the alternative-swapping algorithm to make designs, so flags must be added to the candidate set indicating which candidate can appear in which alternative position in the choice design. In this case, any candidate alternative can appear in any design position. Hence, all flags are constant, and all are set to 1 for every candidate. Eight flag variables, `f1-f8`, are set to 1 for the entire candidate set using the `retain` statement. The `Brand` factor is also named in the `retain` statement. This is just for aesthetic reasons, so that the brand variable gets positioned in the SAS data set after the flag variables and before the `x1-x8` attributes that come in with the `set` statement. The `set` statement reads and concatenates the two candidate sets. The data set options `in=d1` and `in=d2` create two binary variables that are 1 or true when the observation comes from the designated data set and 0 otherwise. With a `set` statement specification like this, when `d1` is 1 then `d2` must be zero and vice versa. Hence, you could just create one of these variables. SAS automatically drops these variables from the output data set.

The remaining steps make copies of the candidates, one copy for each brand that applies. There could be more statements here changing the levels in alternative-specific ways. For example, actual sizes and prices could be substituted for the raw (1, 2, 3, ...) design values, and prices could be assigned differently for the different brands or brand types.

Search the Candidate Set of Alternatives

The following three steps search the full candidate set of alternatives and create designs with 6, 7, and 8 alternatives:

```
* For the next three calls to the choiceff macro:
* Recode model with alternative-specific effects?
* Drop maxiter=1 or increase the value later.
* Consider increasing nsets= later.
;

                                   /* get a design with 6 alternatives     */
%choiceff(data=all,                /* candidate set of alternatives        */
          bestout=b1,              /* name of output design data set       */
                                   /* model with main effects, interaction */
          model=class(brand x1-x6 x1 * x2),
          flags=f1-f6,             /* flag which alt can go where, 6 alts  */
          nsets=20,                /* number of choice sets                */
          maxiter=1,               /* maximum number of designs to make    */
          seed=109,                /* random number seed                   */
          beta=zero)               /* assumed beta vector, Ho: b=0         */

                                   /* get a design with 7 alternatives     */
%choiceff(data=all,                /* candidate set of alternatives        */
          bestout=b2,              /* name of output design data set       */
                                   /* model with main effects, interaction */
          model=class(brand x1-x6 x1 * x2),
          flags=f1-f7,             /* flag which alt can go where, 7 alts  */
          nsets=20,                /* number of choice sets                */
          maxiter=1,               /* maximum number of designs to make    */
          seed=114,                /* random number seed                   */
          beta=zero)               /* assumed beta vector, Ho: b=0         */
```

```
                                    /* get a design with 8 alternatives     */
%choiceff(data=all,                 /* candidate set of alternatives        */
          bestout=b3,               /* name of output design data set       */
                                    /* model with main effects, interaction */
          model=class(brand x1-x6 x1 * x2),
          flags=f1-f8,              /* flag which alt can go where, 8 alts  */
          nsets=20,                 /* number of choice sets                */
          maxiter=1,                /* maximum number of designs to make    */
          seed=121,                 /* random number seed                   */
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

These three designs consist of full choice sets consisting of differing numbers of alternatives. They are further processed in subsequent steps to remove sets with duplicate brands, then they are searched to make the final design. These three steps differ in only two important ways. The `flags=` variable names 6, 7, and 8 flag variables, which means that the steps produce 6, 7, and 8 alternative choice sets. Also, the output data sets with the best designs are all given different names. These steps could have alternative-specific factors coded. Later, when the code is all debugged, you can increase the `maxiter=` value to make more designs from which to choose the best one.

Create a Candidate Set of Choice Sets

The following step combines the three output data sets into one:

```
* Concatenate the three designs, create a new Set variable,
* and flag the three different sizes of choice sets with SetType=1, 2, 3.
* We will use this (with a bit more modification) as a candidate set for
* making the final design.
;
data best(keep=Set Brand SetType AltType x1-x6);
   set b1(in=b1) b2(in=b2) b3(in=b3);
   if set ne lag(set) then newset + 1;
   SetType = b1 + 2 * b2 + 3 * b3;
   AltType = 1 + (brand gt 2);
   set = newset;
   run;
```

Again, the `in=` option is used to flag which observations come from which data set. A new `SetType` variable is created with values of 1 (`b1`) when the observation is of the first type (from the first data set), 2 (2 * `b2`) when the observation is of the from the second data set, and 3 (3 * `b3`) when the observation is of the from the third data set. Only one of the `b1-b3` variables is true or 1 at a time. The type of alternative is also stored here in the variable `AltType`. One other thing is done in this step. Each input data set has its own choice set ID variable, `set`, so this variable starts over at one for each type of choice set. A new `set` variable is created that does not start over at one. Whenever the original `set` variable changes, a new set variable is incremented, and its value is stored in place of the original set variable.

Exclude Choice Sets with Duplicate Brands

There is nothing in the preceding steps that ensures that brands occur only once in each choice set. The following statements identity the choice sets where each brand occurs only once and excludes the choice sets where brands occur more than once:

```
* Extract just the choice sets where no brand appears more than once.
* In other words, if the maximum frequency by set is 1, keep it.
* Start by seeing how often each brand occurs within each set;
;
proc freq data=best noprint; tables set * brand / out=list; run;

* Find the maximum frequency.;
proc means noprint; var count; by set;
   output out=maxes(where=(_stat_ eq 'MAX'));
   run;

* Output the set number if the maximum frequency is one.;
data sets; set; if count = 1; keep set; run;

* Select the sets where the maximum frequency is one.;
data best; merge best sets(in=one); by set; if one; run;

* Report on the set and SetType variables as an error check.;
proc freq; tables set * SetType / list; run;

* Sort the design by brand within set.;
proc sort data=best; by set brand; run;

* Add alternative numbers within set.  Get consecutive set numbers again.;
data best(drop=oldset);
   set best(rename=(set=oldset)); by oldset;
   if first.oldset then do; Alt = 0; Set + 1; end;
   alt + 1;
   call symputx('maxset', set);
   run;

* Display the candidate design.;
proc print; var settype alttype brand x1-x6; by set; id set alt; run;
```

First, PROC FREQ is used to create a list of the number of times each brand occurs in each choice set. This list is stored in a SAS data set. PROC MEANS is used to process this data set and output the maximum brand frequency within each choice set. Next, a data set SETS is created that contains only the choice set numbers where the maximum brand frequency is 1. These are the sets we want. Next, the full candidate set is merged with the list of choice set numbers. Choice sets that are represented in both input data sets are kept, and the rest are deleted. PROC FREQ is run to create a list of observation types within choice set as a check on the results. Some of the results are as follows:

```
                              The FREQ Procedure

                                                   Cumulative    Cumulative
     Set    SetType    Frequency     Percent     Frequency       Percent
     ------------------------------------------------------------------
       1          1            6        1.46             6          1.46
       .
       .
       .
      20          1            6        1.46           120         29.13
      21          2            7        1.70           127         30.83
       .
       .
       .
      40          2            7        1.70           260         63.11
      41          3            8        1.94           268         65.05
       .
       .
       .
      54          3            8        1.94           372         90.29
      56          3            8        1.94           380         92.23
      57          3            8        1.94           388         94.17
      58          3            8        1.94           396         96.12
      59          3            8        1.94           404         98.06
      60          3            8        1.94           412        100.00
```

The full results show that all choice sets in the range 1–20 are of type 1, all choice sets in the range 21–40 are of type 2, all choice sets in the range 41–60 are of type 3. Furthermore, some choice sets (e.g. set 55) have been excluded.

This candidate set of choice sets is still not in the final form for the `%ChoicEff` macro to search. The problem is the `%ChoicEff` macro insists that candidate sets of choice sets must all contain the same number of alternatives. This is not a problem, because a mechanism is in place for dealing with varying numbers of alternatives. Extra (dummy) alternatives are added and given zero weight. This is accomplished in the following steps:

```
* Make the choice set alternative numbers you would have if all
* sets had the maximum 8 alternatives.  We will need this kind of layout
* because ChoicEff assumes all sets have the same alternatives, and uses
* weights when they don't.
;
data frame;
   do Set = 1 to &maxset;
      do Alt = 1 to 8;
         output;
         end;
      end;
   run;

* Add missing observations to the sets with 6 and 7 alternatives.
* Flag alternatives actually there with w = 1 and the rest with w = 0.
;
data all; merge frame best(in=b); by set alt; w = b; run;

proc print; var settype brand alttype x1-x6; by set; id set alt w; run;
```

The first step creates a SAS data set with the choice set and alternative numbers that we need. The number of choice sets comes from a macro variable set in a previous step. This is the number of choice sets that are left after the sets with duplicate brands are excluded. This data set is merged with the actual design and alternatives that come from the actual design are flagged with a weight of one (`w` = 1).

Three of the choice sets are as follows:

```
                        Set                Alt
  Set     Alt     w     Type     Brand     Type     x1     x2     x3     x4     x5     x6

    1      1      1      1          5        2       5      1      1      3      1      3
           2      1      1          6        2       4      4      4      1      2      2
           3      1      1          7        2       3      3      1      2      2      3
           4      1      1          8        2       5      4      2      1      1      1
           5      1      1          9        2       5      2      3      3      1      4
           6      1      1         10        2       5      3      1      3      2      2
           7      0      .          .        .       .      .      .      .      .      .
           8      0      .          .        .       .      .      .      .      .      .

   21      1      1      2          1        1       1      2      2      1      1      3
           2      1      2          2        1       2      3      2      2      1      1
           3      1      2          3        2       5      4      3      3      2      3
           4      1      2          4        2       3      1      3      2      2      4
           5      1      2          6        2       4      2      3      1      2      2
           6      1      2          8        2       2      1      1      2      2      2
           7      1      2          9        2       2      4      4      3      2      1
           8      0      .          .        .       .      .      .      .      .      .
   41    1     1      3       3      2      5     1     1     3     1     3
         2     1      3       4      2      4     1     1     1     2     1
         3     1      3       5      2      5     3     1     3     2     2
         4     1      3       6      2      1     4     4     3     2     4
         5     1      3       7      2      3     4     4     3     1     1
         6     1      3       8      2      2     4     2     1     1     3
         7     1      3       9      2      3     1     3     2     2     4
         8     1      3      10      2      1     2     3     1     1     4
```

You can see that each choice set has 8 data set observations even if it has only 6 or 7 alternatives. This is the form that is needed for making the final design.

Search the Candidate Set of Choice Sets

The following step creates the final design:

```
   * Create a final design with 3 types of choice sets,
   * 12 with 6 alternatives, 12 with 7, and 12 with 8, using weights
   * to ignore the dummy alternatives.
   * The candidate set has all three types of choice sets.
   ;

   %choiceff(data=all,                 /* candidate set of choice sets         */
                                       /* model with main effects, interaction */
            model=class(brand x1-x6 x1 * x2),
            nalts=8,                   /* number of alternatives               */
            weight=w,                  /* weight to ignore dummy alternatives  */
            types=12 12 12,            /* number of each type of set           */
            typevar=settype,           /* choice set types variable            */
            nsets=36,                  /* number of choice sets                */
            maxiter=1,                 /* maximum number of designs to make    */
            seed=396,                  /* random number seed                   */
            beta=zero)                 /* assumed beta vector, Ho: b=0         */
```

The combination of the `typevar=settype` option and the `types=12 12 12` option ensures that each type of choice set appears in the design exactly 12 times. The `weight=` option ensures that only the actual alternatives are used in the design.

The main results from the `%ChoicEff` macro are as follows:

```
                              Final Results

                        Design                  1
                        Choice Sets            36
                        Alternatives            8
                        Parameters             37
                        Maximum Parameters    252
                        D-Efficiency       2.1018
                        D-Error            0.4758
```

```
           Variable                                           Standard
     n       Name       Label            Variance     DF        Error

     1      Brand1      Brand  1          0.72908      1      0.85386
     2      Brand2      Brand  2          0.78231      1      0.88448
     3      Brand3      Brand  3          0.58411      1      0.76427
     4      Brand4      Brand  4          0.56562      1      0.75208
     5      Brand5      Brand  5          0.53984      1      0.73473
     6      Brand6      Brand  6          0.59600      1      0.77201
     7      Brand7      Brand  7          0.58271      1      0.76335
     8      Brand8      Brand  8          0.58995      1      0.76808
     9      Brand9      Brand  9          0.55157      1      0.74268
    10      x11         x1 1              1.61917      1      1.27247
    11      x12         x1 2              1.44374      1      1.20156
    12      x13         x1 3              1.61199      1      1.26964
    13      x14         x1 4              1.55168      1      1.24566
    14      x21         x2 1              1.47910      1      1.21618
    15      x22         x2 2              1.49892      1      1.22430
    16      x23         x2 3              1.58831      1      1.26028
    17      x31         x3 1              0.31441      1      0.56072
    18      x32         x3 2              0.33885      1      0.58211
    19      x33         x3 3              0.34725      1      0.58928
    20      x41         x4 1              0.21259      1      0.46108
    21      x42         x4 2              0.22298      1      0.47220
    22      x51         x5 1              0.12020      1      0.34669
    23      x61         x6 1              0.29928      1      0.54706
    24      x62         x6 2              0.30377      1      0.55115
    25      x63         x6 3              0.35651      1      0.59709
    26      x11x21      x1 1 * x2 1       2.95839      1      1.72000
    27      x11x22      x1 1 * x2 2       3.05417      1      1.74762
    28      x11x23      x1 1 * x2 3       3.05268      1      1.74719
    29      x12x21      x1 2 * x2 1       2.86368      1      1.69224
    30      x12x22      x1 2 * x2 2       2.70056      1      1.64334
    31      x12x23      x1 2 * x2 3       3.13789      1      1.77141
    32      x13x21      x1 3 * x2 1       3.09384      1      1.75893
    33      x13x22      x1 3 * x2 2       2.95469      1      1.71892
    34      x13x23      x1 3 * x2 3       3.01531      1      1.73647
    35      x14x21      x1 4 * x2 1       2.79842      1      1.67285
    36      x14x22      x1 4 * x2 2       3.07207      1      1.75273
    37      x14x23      x1 4 * x2 3       3.36417      1      1.83417
                                                      ==
                                                      37
```

The following steps display the final design and check it for duplicate choice sets:

```
* Display final design.;
proc print;
   var brand settype alttype x1-x6;
   by notsorted set;
   id set alt;
   where w;
   run;

* Check for duplicate choice sets.;
proc freq data=best; tables set; run;
```

Three of the final choice sets are as follows:

```
                           Set    Alt
   Set    Alt    Brand    Type   Type    x1    x2    x3    x4    x5    x6

    11     1        1       1      1      2     3     2     2     1     1
           2        2       1      1      1     2     2     1     1     3
           3        3       1      2      2     2     1     2     1     4
           4        5       1      2      5     3     1     3     2     2
           5        7       1      2      1     1     3     3     2     1
           6        9       1      2      4     4     4     1     2     2

    23     1        1       2      1      2     1     2     3     1     4
           2        2       2      1      1     3     3     1     1     1
           3        3       2      2      2     2     2     1     2     1
           4        4       2      2      3     2     1     2     1     1
           5        6       2      2      1     2     4     2     2     3
           6        7       2      2      3     1     3     2     2     4
           7       10       2      2      4     1     2     3     1     2

    56     1        1       3      1      2     2     3     1     2     4
           2        2       3      1      2     3     3     3     2     3
           3        3       3      2      5     4     2     1     1     1
           4        4       3      2      2     1     1     2     2     2
           5        5       3      2      4     1     2     3     1     2
           6        6       3      2      5     2     2     2     2     2
           7        7       3      2      3     3     4     1     1     2
           8        8       3      2      4     4     2     3     1     1
```

The choice set numbers refer to the input candidate design, so numbers can go up and down. Only the alternatives that are actually used are displayed. The full design shows that different subsets of brands appear in different choice sets and no brand appears more than once in any particular choice set.

The PROC FREQ output (not shown) shows that each alternative occurs equally often (8 times) in the final design. Note that the full design including the dummy alternatives is input to PROC FREQ.

Most designs are not nearly as complicated as this one. However, it is good to know that when complex design problems come up, the tools are there to tackle them.

Making the Candidate Set

The %ChoicEff macro can be used in two ways, either with a candidate set of alternatives or with a candidate set of choice sets. Either way, typically you will use the %MktEx macro to make the candidate set. Before you make the candidate set, you have to decide how big to make it. You can use the %MktRuns macro to get some ideas. Next, based on the information provided by the %MktRuns macro, you make sets of different sizes, try each one, and then see which one works best. Sometimes, bigger is better; other times it is not. It is always the case that a very small candidate set that contains all of the right information for constructing the optimal design is better than a very large candidate set that contains many additional and nonoptimal candidates. However, you typically cannot know how good a candidate set is until you try using it. Typically, you should begin by trying a few iterations using the smallest candidate set that you can reasonably make. Then you should try increasingly bigger sizes, again with just a few iterations. The number of iterations might be on the order of less than ten up to several hundred depending on the problem. At this point, you should not let the %ChoicEff macro run for more than a few minutes. Based on the results, you should pick the size that seems to be working best, and then try more iterations with it. This process is illustrated in the rest of this section.

Candidate Set of Choice Sets

This example uses the %ChoicEff macro to create an efficient choice design from a set of candidate choice sets. The experiment has 3 three-level attributes and three alternatives. First, the %MktRuns macro is run to get suggestions for design sizes. The model specification is 3^9. Later, the design with 9 three-level factors is converted to a set of choice sets with three attributes and three alternatives. The following step produces the design suggestions:

```
%mktruns(3 ** 9)
```

Some of the results are as follows:

```
Saturated      = 19
Full Factorial = 19,683

Some Reasonable                        Cannot Be
   Design Sizes       Violations      Divided By

            27 *               0
            36 *               0
            45 *               0
            54 *               0
            21                36      9
            24                36      9
            30                36      9
            33                36      9
            39                36      9
            42                36      9
            19 S              45      3 9
```

```
* - 100% Efficient design can be made with the MktEx macro.
S - Saturated Design - The smallest design that can be made.
    Note that the saturated design is not one of the
    recommended designs for this problem.  It is shown
    to provide some context for the recommended sizes.
```

Explicitly, these results suggest using the sizes 27, 36, 45, and 54. Extrapolating, the results suggest using sizes that are multiples of powers of 3 beginning with 27. The following statements consider some relevant values:

```
%mktex(3 ** 9, n=27, seed=292)

%mktroll(design=randomized, key=3 3, out=cand)

%choiceff(data=cand,                /* candidate set of choice sets        */
          model=class(x1-x3 / sta), /* model with stdz orthogonal coding   */
          seed=513,                 /* random number seed                  */
          maxiter=5,                /* maximum number of designs to make   */
          options=relative nodups,  /* display relative D-efficiency       */
          nsets=18,                 /* number of choice sets               */
          nalts=3,                  /* number of alternatives              */
          beta=zero)                /* assumed beta vector, Ho: b=0        */

%mktex(3 ** 9, n=36, seed=292)
%mktroll(design=randomized, key=3 3, out=cand)
%choiceff(data=cand, model=class(x1-x3 / sta), seed=513, maxiter=5,
          options=relative nodups, nsets=18, nalts=3, beta=zero)

%mktex(3 ** 9, n=45, seed=292)
%mktroll(design=randomized, key=3 3, out=cand)
%choiceff(data=cand, model=class(x1-x3 / sta), seed=513, maxiter=5,
          options=relative nodups, nsets=18, nalts=3, beta=zero)

%mktex(3 ** 9, n=54, seed=292)
%mktroll(design=randomized, key=3 3, out=cand)
%choiceff(data=cand, model=class(x1-x3 / sta), seed=513, maxiter=5,
          options=relative nodups, nsets=18, nalts=3, beta=zero)

%mktex(3 ** 9, n=3 ** 4, seed=292)
%mktroll(design=randomized, key=3 3, out=cand)
%choiceff(data=cand, model=class(x1-x3 / sta), seed=513, maxiter=5,
          options=relative nodups, nsets=18, nalts=3, beta=zero)

%mktex(3 ** 9, n=3 ** 5, seed=292)
%mktroll(design=randomized, key=3 3, out=cand)
%choiceff(data=cand, model=class(x1-x3 / sta), seed=513, maxiter=5,
          options=relative nodups, nsets=18, nalts=3, beta=zero)
```

```
%mktex(3 ** 9, n=3 ** 6, seed=292)
%mktroll(design=randomized, key=3 3, out=cand)
%choiceff(data=cand, model=class(x1-x3 / sta), seed=513, maxiter=5,
          options=relative nodups, nsets=18, nalts=3, beta=zero)

%mktex(3 ** 9, n=3 ** 7, seed=292)
%mktroll(design=randomized, key=3 3, out=cand)
%choiceff(data=cand, model=class(x1-x3 / sta), seed=513, maxiter=5,
          options=relative nodups, nsets=18, nalts=3, beta=zero)

%mktex(3 ** 9, n=3 ** 8, seed=292)
%mktroll(design=randomized, key=3 3, out=cand)
%choiceff(data=cand, model=class(x1-x3 / sta), seed=513, maxiter=5,
          options=relative nodups, nsets=18, nalts=3, beta=zero)

%mktex(3 ** 9, n=3 ** 9, seed=292)
%mktroll(design=randomized, key=3 3, out=cand)
%choiceff(data=cand, model=class(x1-x3 / sta), seed=513, maxiter=5,
          options=relative nodups, nsets=18, nalts=3, beta=zero)
```

In each case, a candidate set is constructed with nine attributes (one for each of the three attributes for each of the three alternatives), it is rolled into a choice design, then the `%ChoicEff` macro is used to create a choice design from the candidate set of choice sets. More information about these macros can be found elsewhere in this chapter and throughout the examples in the design chapter beginning on page 53 and the “Discrete Choice” chapter beginning on page 285. The results of all of these steps are not shown, but a summary of the resulting *D*-Efficiencies is as follows:

n	*D*-Efficiency
27	13.574097
36	13.818249
45	13.830425
54	14.209785
81	15.218103
243	15.286949
729	16.635262
2,187	18.000000
6,561	17.962771
19,683	17.962771

In this example, with these seeds and this number of iterations, the *D*-efficiency increases with the size of the candidate set up to `n=2187`, then it slightly decreases. *D*-efficiency first increases as the richness of the candidate set increases, then it decreases as the candidate set contains more nonoptimal candidates from which to choose. The results would quite likely be different with different seeds, different numbers of iterations, and different problems. However, this pattern of results is not unusual.

In this problem, the model specification is simple enough that run time is not an issue even with large candidate sets, so you could easily try more than five iterations, particularly with the smaller candidate sets. With other problems, you need to be careful to not make candidate sets that are too big. For more complicated models, candidate sets of several thousand choice sets might be too big.

Examination of the results for `n=2187` (not shown) shows that this design is in fact optimal. Had it not been, you would have run the `%ChoicEff` macro again with candidate set, this time requesting more iterations.

Candidate Set of Alternatives

This section illustrates design creation with a candidate set of alternatives. This section illustrates the fact that a bigger candidate set is not always better. The goal is to make a design with 6 three-level factors in six choice sets each with three alternatives. The following `%MktRuns` macro step suggests candidate set sizes:

```
   %mktruns(3 ** 6)
```

Some of the results are as follows:

```
                    Saturated      = 13
                    Full Factorial = 729

                    Some Reasonable                     Cannot Be
                       Design Sizes       Violations    Divided By

                                 18 *              0
                                 27 *              0
                                 36 *              0
                                 15               15    9
                                 21               15    9
                                 24               15    9
                                 30               15    9
                                 33               15    9
                                 13 S             21    3 9
                                 14               21    3 9

              * - 100% Efficient design can be made with the MktEx macro.
              S - Saturated Design - The smallest design that can be made.
```

Explicitly, these results suggest using the sizes 18, 27, and 36. More generally, the results show that suitable candidate set sizes include multiples of powers of three that are greater than $6(3-1) = 12$ including 18, 27, 36, 54, 72, 81, 108, and so on. The following steps make and search candidate sets of varying sizes:

```
%mktex(3 ** 6, n=18, seed=306)

%choiceff(data=design,               /* candidate set of alternatives       */
          model=class(x1-x6 / sta), /* model with stdz orthogonal coding   */
          maxiter=400,              /* maximum number of designs to make   */
          seed=121,                 /* random number seed                  */
          flags=3,                  /* 3 alternatives, generic candidates  */
          nsets=6,                  /* number of choice sets               */
          options=relative,         /* display relative D-efficiency       */
          beta=zero)                /* assumed beta vector, Ho: b=0        */


%mktex(3 ** 6, n=27, seed=306)
%choiceff(data=design, model=class(x1-x6 / sta), maxiter=400, seed=121,
          flags=3, nsets=6, options=relative, beta=zero)

%mktex(3 ** 6, n=36, seed=306)
%choiceff(data=design, model=class(x1-x6 / sta), maxiter=400, seed=121,
          flags=3, nsets=6, options=relative, beta=zero)

%mktex(3 ** 6, n=54, seed=306)
%choiceff(data=design, model=class(x1-x6 / sta), maxiter=400, seed=121,
          flags=3, nsets=6, options=relative, beta=zero)

%mktex(3 ** 6, n=72, seed=306)
%choiceff(data=design, model=class(x1-x6 / sta), maxiter=400, seed=121,
          flags=3, nsets=6, options=relative, beta=zero)

%mktex(3 ** 6, n=81, seed=306)
%choiceff(data=design, model=class(x1-x6 / sta), maxiter=400, seed=121,
          flags=3, nsets=6, options=relative, beta=zero)

%mktex(3 ** 6, n=108, seed=306)
%choiceff(data=design, model=class(x1-x6 / sta), maxiter=400, seed=121,
          flags=3, nsets=6, options=relative, beta=zero)
```

The results of these steps are not shown, but a summary is as follows:

n	*D*-Efficiency
18	6.000000
27	4.711253
36	4.996099
54	6.000000
72	4.996099
81	4.711253
108	6.000000

The following discussion provides some context for these results. In this problem, the optimal design can be directly constructed, displayed, and evaluated as follows:

```
%mktex(6 3 ** 6, n=18, seed=306)

%mktlab(data=design, vars=Set x1-x6, out=final)

proc print data=final;
   by set;
   id set;
   var x1-x6;
   run;

%choiceff(data=final,                /* candidate set of choice sets         */
          init=final(keep=set),      /* select these sets from candidates    */
          model=class(x1-x6 / sta), /* model with stdz orthogonal coding    */
          nalts=3,                   /* number of alternatives               */
          nsets=6,                   /* number of choice sets                */
          options=relative,          /* display relative D-efficiency        */
          beta=zero)                 /* assumed beta vector, Ho: b=0         */
```

In other words, the 18-run candidate set is the optimal design (D-Efficiency = 6.0 and relative D-Efficiency = 100) when you add a six-level factor and use it as the choice set number. The optimal design is as follows:

```
Set    x1    x2    x3    x4    x5    x6

 1      1     1     1     1     1     1
        2     2     2     2     2     2
        3     3     3     3     3     3

 2      1     1     2     2     3     3
        2     2     3     3     1     1
        3     3     1     1     2     2

 3      1     2     1     3     3     2
        2     3     2     1     1     3
        3     1     3     2     2     1

 4      1     2     3     1     2     3
        2     3     1     2     3     1
        3     1     2     3     1     2

 5      1     3     2     3     2     1
        2     1     3     1     3     2
        3     2     1     2     1     3
```

```
         6        1      3      3      2      1      2
                  2      1      1      3      2      3
                  3      2      2      1      3      1
```

It has a cyclic structure where the second and third alternatives are constructed from the previous alternative by adding 1 (mod 3).[*] This is discussed in the section beginning on page 102. Both the optimal choice design and the 18-run orthogonal array are made using a combinatorial algorithm by developing a 6×6 difference scheme of order 3. The `%ChoicEff` macro does not know that, of course, but it does succeed in sorting the 18-run candidate set into the optimal choice design using its modified-Fedorov search algorithm. The macro also succeeds in constructing the optimal design from candidate sets of size 18, 54 and 108, but not from candidates of size 27, 36, 72, or 81. The right structure is either not in those candidate sets, or the candidate sets are large enough that the optimal design is not found. The former explanation is probably the correct one in this case. For most real-life applications, an optimal design cannot be constructed by combinatorial means, and you do not know which candidate set is best. Usually, the best you can do is try multiple candidate sets and see which one works best.

The rest of this section is optional and can be skipped by all but the most interested readers. The next section starts on page 946. It is interesting to explore the reasons why the optimal design can be found in a candidate set of 18 runs, but not in one of 36 or 72 runs. This is discussed next, but the full combinatorial details are not discussed; you will have to take some aspects of this discussion on faith. Orthogonal arrays have many different underlying constructions. As was mentioned previously, the design in 18 runs is made by developing a 6×6 difference scheme of order 3, which is also the optimal strategy for constructing this choice design. The designs in 27 and 81 runs are made from 9×9 and 27×27 difference scheme of order 3, respectively. This is not the optimal strategy for constructing this choice design. The arrays in 36, 72, and 108 runs can potentially be made in many different ways. You can see this by having the `%MktOrth` macro list all known orthogonal arrays in 36 runs that have at least 6 three-level factors. Note, however, that the words "all known" in the preceding section need some qualification. This list does not include duplicate or inferior designs that are generated with alternative lineages. This point is more fully explained throughout the rest of this section.

The following statements generate the list of designs:

```
   %mktorth(range=n=36, options=lineage)

   proc print data=mktdeslev; var lineage; where x3 ge 6; run;
```

[*]More precisely, since these numbers are based on one instead of zero, the operation is: (x mod 3) + 1.

The results are as follows:

```
         Obs                                   Lineage

           9    36 ** 1 : 36 ** 1 > 3 ** 12 12 ** 1 : 12 ** 1 > 2 ** 11
          10    36 ** 1 : 36 ** 1 > 2 ** 10 3 ** 8 6 ** 1 (parent)
          14    36 ** 1 : 36 ** 1 > 3 ** 12 12 ** 1 : 12 ** 1 > 2 ** 4 3 ** 1
          16    36 ** 1 : 36 ** 1 > 2 ** 3 3 ** 9 6 ** 1 (parent)
          18    36 ** 1 : 36 ** 1 > 3 ** 12 12 ** 1 : 12 ** 1 > 2 ** 2 6 ** 1
          21    36 ** 1 : 36 ** 1 > 3 ** 7 6 ** 3 : 6 ** 1 > 2 ** 1 3 ** 1
          23    36 ** 1 : 36 ** 1 > 3 ** 12 12 ** 1 : 12 ** 1 > 3 ** 1 4 ** 1
          24    36 ** 1 : 36 ** 1 > 3 ** 12 12 ** 1 (parent)
          25    36 ** 1 : 36 ** 1 > 3 ** 7 6 ** 3 (parent)
```

The design lineage is a set of instructions for making a design with smaller-level factors from a design with higher-level factors. For example, the first design is $3^{12}2^{11}$, which is made by replacing a single 36-level factor with $3^{12}12^1$ and then by replacing 12^1 with 2^{11}. There are four underlying parent designs capable of making at least 6 three-level factors in 36 runs and five more child designs. The `%MktEx` macro is capable of using any one of them. You can find out which one the `%MktEx` macro actually uses by default by specifying `options=lineage`, as follows:

```
%mktex(3 ** 6,                     /* 6 three-level factors              */
       n=36,                       /* 36 runs                            */
       seed=306,                   /* random number seed                 */
       options=lineage)            /* display OA construction instructions */
```

The preceding step displays the following lineage:

```
Design Lineage:
36 ** 1 : 36 ** 1 > 3 ** 12 12 ** 1 : 12 ** 1 > 2 ** 11
```

The macro uses the first lineage in the list. None of these involve a 6×6 difference scheme of order 3. A design in 36 runs is twice as big as a design in 18 runs, but it typically will not consist of replicates of the smaller design. Its construction is often quite different. You cannot expect a design of size cn (where c is an integer greater than 1) to contain all of the optimal information found in a design of size n. Sometimes it might, as in the case of the designs in 54 and 108 runs. Other times, the larger designs will not work as well.

The rest of this section explores one more bit of esoteric information that is related to this example. While this information is interesting to those interested in the finer points of choice design, it is not something you should ever need to do in practice. You can control the method that the `%MktEx` macro uses to make the design by explicitly controlling the design catalog that `%MktEx` otherwise creates automatically. You can use the `%MktOrth` macro to make the full catalog of n-run designs and then feed the lineage for just the desired design into the `%MktEx` macro.

The following steps create the design:

```
%mktorth(range=n=36, options=lineage dups)

data lev;
   set mktdeslev;
   where lineage ? '2 ** 2 18' and lineage ? '3 ** 6 6' and
         not (lineage ? ': 6');
   run;

%mktex(3 ** 6,                     /* 6 three-level factors              */
      n=36,                        /* 36 runs                            */
      seed=306,                    /* random number seed                 */
      options=lineage,             /* display OA construction instructions */
      cat=lev)                     /* OA catalog comes from lev data set   */

%choiceff(data=design,             /* candidate set of alternatives      */
         model=class(x1-x6 / sta), /* model with stdz orthogonal coding  */
         maxiter=400,              /* maximum number of designs to make  */
         seed=121,                 /* random number seed                 */
         flags=3,                  /* 3 alternatives, generic candidates */
         nsets=6,                  /* number of choice sets              */
         options=relative,         /* display relative D-efficiency      */
         beta=zero)                /* assumed beta vector, Ho: b=0       */
```

The `dups` option in the `%MktOrth` macro is used to include duplicate and inferior designs in the catalog. These are designs that are normally removed from the catalog. An inferior design has only a subset of the factors that are available in a competing design. By default (without the `dups` option), the `%MktOrth` macro lists 26 designs with 36 runs. With the `dups` option, the number increases to 61. In this case, the design of interest is $2^2 3^6 6^1$ in 36 runs, where $3^6 6^1$ is constructed from an 18-level factor. Normally, it is removed from the catalog since it has fewer two-level and three-level factors than $2^{10} 3^8 6^1$. While it is inferior as an orthogonal array, it is a better candidate set than the default 36-run design, but less good than the 18-run design (since it is twice as big as it needs to be).

The `where` clause in the DATA step selects a design with 2 two-level factors and an 18-level factor in its lineage, 6 three-level factors and a six-level factor in its lineage, and no mention of expanding the six-level factor. This selects just the one design that we are interested in. The new catalog, with just the design of interest, is input to the `%MktEx` macro using the `cat=` (catalog input data set) option. The resulting design is then prepared as a candidate set and is input the `%ChoicEff` macro. With this candidate set, the `%ChoicEff` macro finds an optimal design.

Initial Designs and Evaluating a Design

The `%ChoicEff` macro can be used to either search for a design or to evaluate an existing design. This section discusses using the `%ChoicEff` macro to evaluate a design. In all cases, whether you are searching for a design or evaluating a design, you must provide the `%ChoicEff` macro with a candidate set. It might be the candidate set that was used to construct the design or it might simply be the design itself. When you evaluate a design, you must also provide information about how to construct the initial design from the candidate set. This can happen in one of several ways:

- If you have a design you want to evaluate in a SAS data set (say `Final`), and it has the choice set number (say `Set`), use the options `data=Final, init=Final(keep=set), intiter=0` to evaluate the design. This approach uses the choice set numbers in the `init=` data set to select the matching choice sets (the entire design) from the `data=` data set. It performs zero internal iterations. You might want to use this approach when you construct a design using a method other than the `%ChoicEff` macro.

- If you have a design you want to evaluate in a SAS data set (say `Design`), and it does not have the choice set number, but it has factors of say `x1-x6`, use the options `data=Design, init=Design, initvars=x1-x6, intiter=0` to evaluate the design. This approach uses the initial variables in the `init=` data set to select the matching observations (the entire design) in the `data=` data set. It performs zero internal iterations. You might want to use this approach when you are given a design rather than processing it further to add a choice set variable.

- If you have a design you want to evaluate in a SAS data set (say `Best`) that might have been constructed using the `%ChoicEff` macro using the alternative swapping algorithm, and it has a variable (say `Index`) that contains the alternative number from the candidate set of alternatives (say `Cand`) used to make the design, use the options `data=Cand, init=Best(keep=index), intiter=0` to evaluate the design. This approach uses the `Index` variable in the `init=` data set to select the matching alternatives in the `data=` data set. It performs zero internal iterations. You might want to use this approach when you found a design using the `%ChoicEff` macro, and now you want to evaluate it with other options or codings.

- If you have a design you want to evaluate in a SAS data set (say `Best`) that might have been constructed using the `%ChoicEff` macro using the choice set swapping algorithm, and it has a variable (say `Set`) that contains the alternative number from the candidate set of choice sets (say `Cand`) used to make the design, use the options `data=Cand, init=Best(keep=set), intiter=0` to evaluate the design. This approach uses the `Set` variable in the `init=` data set to select the matching choice sets in the `data=` data set. It performs zero internal iterations. You might want to use this approach when you found a design using the `%ChoicEff` macro, and now you want to evaluate it with other options or codings.

With the `intiter=0` option specified the design is evaluated. If you leave this option out, the design is used as an initial design, and the `%ChoicEff` macro will try to iterate to improve it.

The following example constructs a design with the choice set number and evaluates it:

```
%mktex(5 ** 6, n=25)

%mktlab(data=design, vars=Set x1-x5)

                                 /* evaluate design                        */
%choiceff(data=final,            /* candidate set of choice sets           */
          init=final(keep=set),  /* select these sets from candidates      */
          intiter=0,             /* evaluate without internal iterations */
          model=class(x1-x5 / sta), /* model with stdzd orthogonal coding   */
          nsets=5,               /* 5 choice sets                          */
          nalts=5,               /* 5 alternatives per set                 */
          options=relative,      /* display relative D-efficiency          */
          beta=zero)             /* assumed beta vector, Ho: b=0           */
```

Some of the evaluation results are as follows:

```
                              Final Results

                         Design                  1
                         Choice Sets             5
                         Alternatives            5
                         Parameters             20
                         Maximum Parameters     20
                         D-Efficiency       5.0000
                         Relative D-Eff   100.0000
                         D-Error            0.2000
                         1 / Choice Sets    0.2000
```

This design is 100% *D*-efficient.

The following example constructs a design without the choice set number and evaluates it:

```
%mktex(3 ** 6 6, n=18, options=nosort)

data design(keep=x1-x6);         /* There are easier ways to make this   */
   set design(obs=6);            /* design.  This example is just for    */
   array x[6];                   /* illustration.                        */
   output;
   do i = 1 to 6; x[i] = mod(x[i], 3) + 1; end;
   output;
   do i = 1 to 6; x[i] = mod(x[i], 3) + 1; end;
   output;
   run;
```

```
%choiceff(data=design,              /* candidate set of choice sets         */
          init=design,              /* initial design                       */
          initvars=x1-x6,           /* factors in the initial design        */
          intiter=0,                /* evaluate without internal iterations */
          model=class(x1-x6 / sta), /* model with stdzd orthogonal coding   */
          nsets=6,                  /* 6 choice sets                        */
          nalts=3,                  /* 3 alternatives per set               */
          options=relative,         /* display relative D-efficiency        */
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

Some of the evaluation results are as follows:

```
                                  Final Results

                            Design                   1
                            Choice Sets              6
                            Alternatives             3
                            Parameters              12
                            Maximum Parameters      12
                            D-Efficiency        6.0000
                            Relative D-Eff    100.0000
                            D-Error             0.1667
                            1 / Choice Sets     0.1667
```

This design is 100% *D*-efficient.

The following example constructs a design using the alternative-swapping algorithm and then evaluates it using the standardize orthogonal contrast coding:

```
%mktex(3 ** 3, n=27, seed=238)

                                    /* search for a design                  */
%choiceff(data=randomized,          /* candidate set of alternatives        */
          model=class(x1-x3),       /* main effects with ref cell coding    */
          nsets=3,                  /* number of choice sets                */
          flags=3,                  /* 3 alternatives, generic candidates   */
          seed=382,                 /* random number seed                   */
          beta=zero)                /* assumed beta vector, Ho: b=0         */

                                    /* evaluate design                      */
%choiceff(data=randomized,          /* candidate set of alternatives        */
          init=best(keep=index),    /* select these alts from candidates    */
          intiter=0,                /* evaluate without internal iterations */
          model=class(x1-x3 / sta), /* model with stdz orthogonal coding    */
          nsets=3,                  /* number of choice sets                */
          flags=3,                  /* 3 alternatives, generic candidates   */
          options=relative,         /* display relative D-efficiency        */
          beta=zero)                /* assumed beta vector, Ho: b=0         */
```

Some of the design creation results are as follows:

```
                    Final Results

              Design                  2
              Choice Sets             3
              Alternatives            3
              Parameters              6
              Maximum Parameters      6
              D-Efficiency       0.5774
              D-Error            1.7321
```

Some of the evaluation results are as follows:

```
                    Final Results

              Design                  1
              Choice Sets             3
              Alternatives            3
              Parameters              6
              Maximum Parameters      6
              D-Efficiency       3.0000
              Relative D-Eff   100.0000
              D-Error            0.3333
              1 / Choice Sets    0.3333
```

With the standardized orthogonal contrast coding and the relative *D*-efficiency displayed, it is clear that this design is 100% *D*-efficient. This was not as clear when the design was created without these options.

The following example constructs a design using the set-swapping algorithm and then evaluates it using the standardize orthogonal contrast coding:

```
%mktex(2 ** 6, n=64)

%mktroll(design=design, key=2 3, out=cand)

                                    /* search for a design                 */
%choiceff(data=cand,                /* candidate set of choice sets        */
          model=class(x1-x3),       /* main effects with ref cell coding   */
          nsets=8,                  /* number of choice sets               */
          nalts=2,                  /* number of alternatives              */
          seed=151,                 /* random number seed                  */
          beta=zero)                /* assumed beta vector, Ho: b=0        */

                                    /* evaluate design                     */
%choiceff(data=cand,                /* candidate set of choice sets        */
          init=best(keep=set),      /* select these sets from candidates   */
          intiter=0,                /* evaluate without internal iterations */
          model=class(x1-x3 / sta), /* model with stdzd orthogonal coding  */
          nsets=8,                  /* number of choice sets               */
          nalts=2,                  /* number of alternatives              */
          options=relative,         /* display relative D-efficiency       */
          beta=zero)                /* assumed beta vector, Ho: b=0        */
```

Some of the design creation results are as follows:

```
                              Final Results

                        Design                   1
                        Choice Sets              8
                        Alternatives             2
                        Parameters               3
                        Maximum Parameters       8
                        D-Efficiency        2.0000
                        D-Error             0.5000
```

Some of the evaluation results are as follows:

```
                         Final Results

                   Design                   1
                   Choice Sets              8
                   Alternatives             2
                   Parameters               3
                   Maximum Parameters       8
                   D-Efficiency        8.0000
                   Relative D-Eff    100.0000
                   D-Error             0.1250
                   1 / Choice Sets     0.1250
```

With the standardized orthogonal contrast coding and the relative *D*-efficiency displayed, it is clear that this design is 100% *D*-efficient. This was not as clear when the design was created without these options.

Partial-Profile Designs

The following steps create and evaluate an optimal partial profile design where 4 of 16 attributes vary in each choice set:

```
%mktex(3 ** 4 6, n=18)

proc sort data=design out=design(drop=x5); by x1 x5; run;

%mktbibd(b=20, t=16, k=4, seed=104, out=b)

%mktppro(ibd=b, print=f p)

%choiceff(data=chdes,                /* candidate set of choice sets        */
          init=chdes(keep=set),      /* select these sets from candidates   */
          intiter=0,                 /* no iterations, just evaluate        */
          model=class(x1-x6 / sta),  /* model with stdz orthogonal coding   */
          nsets=120,                 /* number of choice sets               */
          nalts=3,                   /* number of alternatives              */
          rscale=partial=4 of 16,    /* partial profiles, 4 of 16 vary      */
          beta=zero)                 /* assumed beta vector, Ho: b=0        */
```

The raw *D*-efficiency and the relative *D*-efficiency (scaled to range from 0 to 100) are as follows:

```
                              Final Results

                        Design                    1
                        Choice Sets             120
                        Alternatives              3
                        Parameters               12
                        Maximum Parameters      240
                        D-Efficiency        30.0000
                        Relative D-Eff     100.0000
                        D-Error              0.0333
                        1 / Choice Sets    0.008333
```

In this case, since 4 of 16 attributes vary, the maximum *D*-efficiency is not the number of choice sets (120), it is 4/16 times the number of choice sets (30).

Other Uses of the RSCALE=PARTIAL= Option

The `rscale=partial=` option can also be used with constant alternatives. The following example creates and displays a generic design with a constant alternative:

```
%mktex(6 3 ** 6, n=18, seed=104);

%mktlab(data=randomized, vars=Set x1-x6)

proc sort data=final; by set; run;

data chdes;
   set final;
   by set;
   output;
   if last.set then do;
      x1 = .; x2 = .; x3 = .;
      x4 = .; x5 = .; x6 = .;
      output;
      end;
   run;

proc print; by set; id set; run;
```

The first step makes an orthogonal array. The second step converts the six-level factor to the choice set number. The third step sorts the design into choice sets. The fourth step adds a constant alternative after the last alternative in each choice set. The fifth step displays the design.

The results are as follows:

```
Set    x1    x2    x3    x4    x5    x6

  1     3     3     2     3     3     3
        1     2     1     2     2     2
        2     1     3     1     1     1
        .     .     .     .     .     .

  2     1     3     1     1     1     3
        2     2     3     3     3     2
        3     1     2     2     2     1
        .     .     .     .     .     .

  3     3     2     1     1     3     1
        1     1     3     3     2     3
        2     3     2     2     1     2
        .     .     .     .     .     .

  4     1     1     2     1     3     2
        2     3     1     3     2     1
        3     2     3     2     1     3
        .     .     .     .     .     .

  5     1     2     2     3     1     1
        2     1     1     2     3     3
        3     3     3     1     2     2
        .     .     .     .     .     .

  6     3     1     1     3     1     2
        1     3     3     2     3     1
        2     2     2     1     2     3
        .     .     .     .     .     .
```

The following step evaluates the design:

```
%choiceff(data=chdes,               /* candidate set of choice sets        */
          init=chdes(keep=set),     /* select these sets from candidates   */
          intiter=0,                /* no iterations, just evaluate        */
          model=class(x1-x6 / sta), /* model with stdzd orthogonal coding  */
          nsets=6,                  /* 6 choice sets                       */
          nalts=4,                  /* number of alternatives              */
          rscale=partial=3 of 4,    /* relative D-eff, 3 of 4 attrs vary   */
          beta=zero)                /* assumed beta vector, Ho: b=0        */
```

The option `rscale=partial=3 of 4` is specified since 3 of 4 attributes vary in each choice set. The raw *D*-efficiency and the relative *D*-efficiency (scaled to range from 0 to 100) are as follows:

```
                              Final Results

                        Design                   1
                        Choice Sets              6
                        Alternatives             4
                        Parameters              12
                        Maximum Parameters      18
                        D-Efficiency        4.5000
                        Relative D-Eff    100.0000
                        D-Error             0.2222
                        1 / Choice Sets     0.1667
```

The variances and standard errors are as follows:

```
           Variable                                        Standard
      n      Name      Label     Variance     DF         Error

      1      x11       x1 1       0.22222      1       0.47140
      2      x12       x1 2       0.22222      1       0.47140
      3      x21       x2 1       0.22222      1       0.47140
      4      x22       x2 2       0.22222      1       0.47140
      5      x31       x3 1       0.22222      1       0.47140
      6      x32       x3 2       0.22222      1       0.47140
      7      x41       x4 1       0.22222      1       0.47140
      8      x42       x4 2       0.22222      1       0.47140
      9      x51       x5 1       0.22222      1       0.47140
     10      x52       x5 2       0.22222      1       0.47140
     11      x61       x6 1       0.22222      1       0.47140
     12      x62       x6 2       0.22222      1       0.47140
                                              ==
                                              12
```

The following step displays the variance matrix:

```
proc format;
   value zer -1e-12 - 1e-12 = ' 0   ';
   run;

proc print data=bestcov label;
   title 'Variance-Covariance Matrix';
   id __label;
   label __label = '00'x;
   var x:;
   format _numeric_ zer5.2;
   run;
title;
```

The results are as follows:

```
                          Variance-Covariance Matrix

           x1 1  x1 2  x2 1  x2 2  x3 1  x3 2  x4 1  x4 2  x5 1  x5 2  x6 1  x6 2

    x1 1  0.22  0     0     0     0     0     0     0     0     0     0     0
    x1 2  0     0.22  0     0     0     0     0     0     0     0     0     0
    x2 1  0     0     0.22  0     0     0     0     0     0     0     0     0
    x2 2  0     0     0     0.22  0     0     0     0     0     0     0     0
    x3 1  0     0     0     0     0.22  0     0     0     0     0     0     0
    x3 2  0     0     0     0     0     0.22  0     0     0     0     0     0
    x4 1  0     0     0     0     0     0     0.22  0     0     0     0     0
    x4 2  0     0     0     0     0     0     0     0.22  0     0     0     0
    x5 1  0     0     0     0     0     0     0     0     0.22  0     0     0
    x5 2  0     0     0     0     0     0     0     0     0     0.22  0     0
    x6 1  0     0     0     0     0     0     0     0     0     0     0.22  0
    x6 2  0     0     0     0     0     0     0     0     0     0     0     0.22
```

This is an optimal design with 100% relative *D*-efficiency and a diagonal variance matrix.

The following steps search for a design for this problem with a computerized search rather than a direct construction from an orthogonal array:

```
%mktex(3 ** 6, n=729, seed=104);

data cand;
   retain f1-f4 0;
   if _n_ = 1 then do;
      f4 = 1; output; f1 = 1; f2 = 1; f3 = 1; f4 = 0;
      end;
   set randomized;
   output;
   run;

proc print data=cand; run;

%choiceff(data=cand,                /* candidate set of alternatives        */
          model=class(x1-x6 / sta), /* model with stdzd orthogonal coding   */
          flags=f1-f4,              /* flag which alts go where             */
          nsets=6,                  /* 6 choice sets                        */
          maxiter=30,               /* maximum designs to make              */
          rscale=partial=3 of 4,    /* relative D-eff, 3 of 4 attrs vary    */
          seed=104,                 /* random number seed                   */
          beta=zero)                /* assumed beta vector, Ho: b=0         */


proc print data=bestcov label;
   title 'Variance-Covariance Matrix';
   id __label;
   label __label = '00'x;
   var x:;
   format _numeric_ zer5.2;
   run;
title;
```

Part of the candidate set of alternatives is as follows:

```
           Obs    f1    f2    f3    f4    x1    x2    x3    x4    x5    x6

             1     0     0     0     1     .     .     .     .     .     .
             2     1     1     1     0     3     2     3     2     1     1
             3     1     1     1     0     3     3     1     3     3     3
             4     1     1     1     0     1     2     3     1     1     1
             5     1     1     1     0     2     1     1     2     1     2
             6     1     1     1     0     1     2     1     1     3     2
             7     1     1     1     0     2     3     1     3     3     3
             8     1     1     1     0     2     3     3     2     3     1
             9     1     1     1     0     3     1     3     1     2     1
```

```
      .
      .
      .
    728     1     1     1     0     2     3     3     3     3     2
    729     1     1     1     0     3     3     3     1     1     1
    730     1     1     1     0     2     2     1     3     1     1
```

The first candidate provides the constant alternative for each choice set, and the remaining candidates provide the first through third alternatives.

The raw *D*-efficiency, the relative *D*-efficiency (scaled to range from 0 to 100), and the variances and standard errors are as follows:

```
                               Final Results

                         Design                   21
                         Choice Sets               6
                         Alternatives              4
                         Parameters               12
                         Maximum Parameters       18
                         D-Efficiency         4.1425
                         Relative D-Eff      92.0562
                         D-Error              0.2414
                         1 / Choice Sets      0.1667

                 Variable                                     Standard
            n      Name      Label     Variance     DF         Error

            1      x11       x1 1       0.24444      1       0.49441
            2      x12       x1 2       0.28889      1       0.53748
            3      x21       x2 1       0.24444      1       0.49441
            4      x22       x2 2       0.28889      1       0.53748
            5      x31       x3 1       0.23611      1       0.48591
            6      x32       x3 2       0.26389      1       0.51370
            7      x41       x4 1       0.31111      1       0.55777
            8      x42       x4 2       0.22222      1       0.47140
            9      x51       x5 1       0.27778      1       0.52705
           10      x52       x5 2       0.22222      1       0.47140
           11      x61       x6 1       0.24444      1       0.49441
           12      x62       x6 2       0.28889      1       0.53748
                                                    ==
                                                    12
```

The variance matrix is as follows:

```
                       Variance-Covariance Matrix

         x1 1  x1 2  x2 1  x2 2  x3 1  x3 2  x4 1  x4 2  x5 1  x5 2  x6 1  x6 2

    x1 1  0.24 -0.04  0.02 -0.04  0     0    -0.07  0     0     0    -0.03 -0.06
    x1 2 -0.04  0.29 -0.04  0.07  0     0     0.12  0     0     0     0.06  0.10
    x2 1  0.02 -0.04  0.24 -0.04  0     0    -0.07  0     0     0    -0.03 -0.06
    x2 2 -0.04  0.07 -0.04  0.29  0     0     0.12  0     0     0     0.06  0.10
    x3 1  0     0     0     0     0.24  0.02  0     0     0.03  0     0     0
    x3 2  0     0     0     0     0.02  0.26  0     0     0.05  0     0     0
    x4 1 -0.07  0.12 -0.07  0.12  0     0     0.31  0     0     0     0.04  0.08
    x4 2  0     0     0     0     0     0     0     0.22  0     0     0     0
    x5 1  0     0     0     0     0.03  0.05  0     0     0.28  0     0     0
    x5 2  0     0     0     0     0     0     0     0     0     0.22  0     0
    x6 1 -0.03  0.06 -0.03  0.06  0     0     0.04  0     0     0     0.24  0.04
    x6 2 -0.06  0.10 -0.06  0.10  0     0     0.08  0     0     0     0.04  0.29
```

This problem is large enough that it is hard to find the optimal design with a computerized search. Hence, the relative *D*-efficiency is 92.0562 (out of 100), most variances are larger than 0.22, and some covariances are larger than zero.

Optimal Alternative-Specific Design

The following steps create and evaluate an optimal design for a choice model with alternative-specific effects, three brands (and hence three alternatives), 27 choice sets, and 4 three-level attributes in addition to brand:

```
   %mktex(3 ** 12, n=27, seed=104)

   %mktkey(3 4)

   data key; Brand = scan('A B C', _n_); set key; run;

   %mktroll(design=randomized, key=key, out=rolled, alt=brand)
```

```
%choiceff(data=rolled,              /* candidate set of choice sets         */
          init=rolled(keep=set),    /* select these sets from candidates    */
          intiter=0,                /* no iterations, just evaluate         */
          model=class(brand / sta)  /* brand effects                        */
                class(brand * x1    /* alternative-specific effects x1      */
                      brand * x2    /* alternative-specific effects x2      */
                      brand * x3    /* alternative-specific effects x3      */
                      brand * x4 /  /* alternative-specific effects x4      */
                      sta zero=' '),/* std ortho coding, use all brands     */
          nalts=3,                  /* number of alternatives               */
          nsets=27,                 /* number of choice sets                */
          rscale=alt,               /* alt-specific design efficiency scale */
          beta=zero)                /* assumed beta vector, Ho: b=0         */

proc print data=bestcov label;
   title 'Variance-Covariance Matrix';
   id __label;
   label __label = '00'x;
   var B:;
   format _numeric_ zer5.2;
   run;
title;
```

The raw *D*-efficiency and the relative *D*-efficiency (scaled to range from 0 to 100) are as follows:

```
                              Final Results

                        Design                    1
                        Choice Sets              27
                        Alternatives              3
                        Parameters               26
                        Maximum Parameters       54
                        D-Efficiency         6.7359
                        Relative D-Eff     100.0000
                        D-Error              0.1485
                        1 / Choice Sets      0.0370
```

The variances and standard errors are as follows:

```
      Variable                                               Standard
 n    Name          Label                 Variance     DF      Error

 1    BrandA        Brand A                0.03704      1      0.19245
 2    BrandB        Brand B                0.03704      1      0.19245
 3    BrandAx11     Brand A * x1 1         0.16667      1      0.40825
 4    BrandAx12     Brand A * x1 2         0.16667      1      0.40825
 5    BrandBx11     Brand B * x1 1         0.16667      1      0.40825
 6    BrandBx12     Brand B * x1 2         0.16667      1      0.40825
 7    BrandCx11     Brand C * x1 1         0.16667      1      0.40825
 8    BrandCx12     Brand C * x1 2         0.16667      1      0.40825
 9    BrandAx21     Brand A * x2 1         0.16667      1      0.40825
10    BrandAx22     Brand A * x2 2         0.16667      1      0.40825
11    BrandBx21     Brand B * x2 1         0.16667      1      0.40825
12    BrandBx22     Brand B * x2 2         0.16667      1      0.40825
13    BrandCx21     Brand C * x2 1         0.16667      1      0.40825
14    BrandCx22     Brand C * x2 2         0.16667      1      0.40825
15    BrandAx31     Brand A * x3 1         0.16667      1      0.40825
16    BrandAx32     Brand A * x3 2         0.16667      1      0.40825
17    BrandBx31     Brand B * x3 1         0.16667      1      0.40825
18    BrandBx32     Brand B * x3 2         0.16667      1      0.40825
19    BrandCx31     Brand C * x3 1         0.16667      1      0.40825
20    BrandCx32     Brand C * x3 2         0.16667      1      0.40825
21    BrandAx41     Brand A * x4 1         0.16667      1      0.40825
22    BrandAx42     Brand A * x4 2         0.16667      1      0.40825
23    BrandBx41     Brand B * x4 1         0.16667      1      0.40825
24    BrandBx42     Brand B * x4 2         0.16667      1      0.40825
25    BrandCx41     Brand C * x4 1         0.16667      1      0.40825
26    BrandCx42     Brand C * x4 2         0.16667      1      0.40825
                                                       ==
                                                       26
```

The variance matrix (not shown) is diagonal. The brand effects have variances equal to one over the number of choice sets (just like in the optimal generic designs). The alternative-specific effects (with 3 alternatives and 27 choice sets) all have variances equal to $3^2/(3-1)/27 = 1/6$. The ratio 1/6 is the inverse of 2/9 of the number of choice sets. With an alternative-specific design with three alternatives, 1/3 of the rows in the coded design have information. Furthermore, $(3-1)/3 = 2/3$ of the rows in any choice design contribute information to the variance matrix. The resulting product is $(1/3) \times (2/3) = 2/9$.

The determinant of the variance matrix can be decomposed into the product of the determinant of the variance submatrix for the brand effects and the determinant of the variance submatrix for the alternative-specific effects (since the off diagonal elements are zero). This determinant (with 2 and 24 parameters in each submatrix) is $27^2 \times (27 \times 2/9)^{24}$. The *D*-efficiency is the 26*th* root of this product since there are 26 parameters.

The following step computes and displays the maximum *D*-efficiency:

```
data _null_;
   sets  = 27;
   alts  = 3;
   m     = alts - 1;
   parms = m + alts * 4 * (3 - 1);
   det1  = sets ** m;
   det2  = (sets * (m / (alts ** 2))) ** (parms - m);
   scale = (det1 * det2) ** (1 / parms);
   put scale=;
   run;
```

The result is "scale=6.7359424316", which matches the raw *D*-efficiency in the final results table and is used as the scaling factor to get the relative *D*-efficiency.

The following steps illustrate this technique with an alternative-specific design created by an orthogonal array with 6-, 4-, 3-, and 2-level factors:

```
%mktex(6 3 2 2 4 4  6 3 2 2 4 4
       6 3 2 2 4 4  6 3 2 2 4 4  6 3 2 2 4 4, n=288)

%mktkey(5 6)

data key; Brand = scan('A B C D E F', _n_); set key; run;

%mktroll(design=randomized, key=key, out=rolled, alt=brand)

%choiceff(data=rolled,              /* candidate set of choice sets        */
          init=rolled(keep=set),    /* select these sets from candidates   */
          intiter=0,                /* no iterations, just evaluate        */
          model=class(brand / sta)  /* brand effects                       */
                class(brand * x1    /* alternative-specific effects x1     */
                      brand * x2    /* alternative-specific effects x2     */
                      brand * x3    /* alternative-specific effects x3     */
                      brand * x4    /* alternative-specific effects x4     */
                      brand * x5    /* alternative-specific effects x5     */
                      brand * x6 /  /* alternative-specific effects x6     */
                      sta zero=' '),/* std ortho coding, use all brands    */
          nalts=5,                  /* number of alternatives              */
          nsets=288,                /* number of choice sets               */
          rscale=alt,               /* alt-specific design efficiency scale */
          beta=zero)                /* assumed beta vector, Ho: b=0        */
```

```
proc print data=bestcov label;
   title 'Variance-Covariance Matrix';
   id __label;
   label __label = '00'x;
   var B:;
   format _numeric_ zer5.2;
   run;
title;

proc iml;
   use bestcov(drop=__:); read all into x;
   x = shape(x, 1)`;
   create _cov from x[colname='Covariance']; append from x;
   quit;

proc freq; tables Covariance; format covariance zer5.; run;

data _null_;
   sets  = 288;
   alts  = 5;
   m     = alts - 1;
   parms = m + alts * (6 + 3 + 2 + 2 + 4 + 4 - 6);
   det1  = sets ** m;
   det2  = (sets * (m / (alts ** 2))) ** (parms - m);
   scale = (det1 * det2) ** (1 / parms);
   put scale=;
   run;
```

The raw *D*-efficiency and the relative *D*-efficiency (scaled to range from 0 to 100) are as follows:

```
                          Final Results

                    Design                    1
                    Choice Sets             288
                    Alternatives              5
                    Parameters               79
                    Maximum Parameters     1152
                    D-Efficiency        50.5604
                    Relative D-Eff     100.0000
                    D-Error              0.0198
                    1 / Choice Sets    0.003472
```

The variances and covariances are not shown, but they are summarized in the following listing from PROC FREQ:

```
                              The FREQ Procedure

                                                     Cumulative    Cumulative
          Covariance    Frequency     Percent     Frequency      Percent
          ----------------------------------------------------------------
                   0         6162       98.73          6162        98.73
               0.003            4        0.06          6166        98.80
               0.022           75        1.20          6241       100.00
```

The four variances for the brand effects are 0.003, and the 75 variances for the alternative-specific effects are 0.022. All covariances are zero. The results of the DATA _NULL_ step produce "`scale=50.560364105`", which matches the unscaled efficiency. Using this as a scale factor, the relative *D*-efficiency is 100%. This design is optimal, but is quite large with 288 choice sets. The following step creates a smaller design with a computerized search:

```
%mktex(6 3 2 2 4 4, n=6*3*2*2*4*4)

data des(drop=i);
   retain f1-f5 0;
   array f[5];
   set design;
   do i = 1 to 5;
      Brand = scan('A B C D E', i);
      f[i] = 1; output; f[i] = 0;
      end;
   run;

%choiceff(data=des,                 /* candidate set of alternatives       */
          model=class(brand / sta)  /* brand effects                       */
                class(brand * x1    /* alternative-specific effects x1     */
                      brand * x2    /* alternative-specific effects x2     */
                      brand * x3    /* alternative-specific effects x3     */
                      brand * x4    /* alternative-specific effects x4     */
                      brand * x5    /* alternative-specific effects x5     */
                      brand * x6 /  /* alternative-specific effects x6     */
                      sta zero=' '),/* std ortho coding, use all brands    */
          flags=f1-f5,              /* 5 alternatives, generic candidates  */
          nsets=32,                 /* number of choice sets               */
          maxiter=2,                /* maximum number of designs to make   */
          rscale=alt,               /* alt-specific design efficiency scale */
          seed=104,                 /* random number seed                  */
          beta=zero)                /* assumed beta vector, Ho: b=0        */
```

The raw *D*-efficiency and the relative *D*-efficiency (scaled to range from 0 to 100) are as follows:

```
                              Final Results

                        Design                  7
                        Choice Sets            32
                        Alternatives            5
                        Parameters             79
                        Maximum Parameters    128
                        D-Efficiency       5.0863
                        Relative D-Eff    90.5381
                        D-Error            0.1966
                        1 / Choice Sets    0.0313
```

The relative *D*-efficiency is based on comparing this design to a hypothetical alternative-specific choice design in 32 choice sets with a diagonal variance matrix like the one generated with 288 choice sets.

This next step creates a design with brand effects, alternative-specific price effects, and cross effects:

```
%let sets = %eval(3 ** 5);

%mktex(3 ** 5, n=&sets)

%mktlab(values=1.49 1.99 2.49)

data key;
  input (b p) ($);
  datalines;
1 x1
2 x2
3 x3
4 x4
5 x5
. .
;

%mktroll(design=final, key=key, out=crosscan, alt=b, keep=x1-x5)

data crosscan;
   set crosscan;
   label b  = 'Brand' p = 'Price' x1 = 'Brand 1 Price'
         x2 = 'Brand 2 Price' x3 = 'Brand 3 Price'
         x4 = 'Brand 4 Price' x5 = 'Brand 5 Price';
   run;
```

```
%choiceff(data=crosscan,           /* candidate set of choice sets         */
          init=crosscan(keep=set), /* select these sets from candidates    */
          intiter=0,               /* no iterations, just evaluate         */
          model=class(b            /* brand effects                        */
                b*p / zero=' ')    /* alternative-specific effects         */
                class(b / zero=none)/* cross effects                       */
                * identity(x1-x5),
          drop=B1X1 B2X2 B3X3      /* drop cross effects of brand on self  */
               B4X4 B5X5,
          nsets=&sets,             /* number of choice sets                */
          nalts=6,                 /* number of alternatives               */
          beta=zero)               /* assumed beta vector, Ho: b=0         */
```

This design is constructed from a full-factorial design of the price attributes. The final results table is as follows:

```
                              Final Results

                         Design                  1
                         Choice Sets           243
                         Alternatives            6
                         Parameters             35
                         Maximum Parameters   1215
                         D-Efficiency       6.5104
                         D-Error            0.1536
```

This next step is similar to the previous step, but instead of evaluating the design constructed from the full-factorial design, we use it as a candidate set:

```
%choiceff(data=crosscan,           /* candidate set of choice sets         */
          model=class(b            /* brand effects                        */
                b*p / zero=' ')    /* alternative-specific effects         */
                class(b / zero=none)/* cross effects                       */
                * identity(x1-x5),
          maxiter=10,              /* maximum number of designs to make    */
          drop=B1X1 B2X2 B3X3      /* drop cross effects of brand on self  */
               B4X4 B5X5,
          nsets=&sets,             /* number of choice sets                */
          nalts=6,                 /* number of alternatives               */
          seed=104,                /* random number seed                   */
          beta=zero)               /* assumed beta vector, Ho: b=0         */
```

The final results table is as follows:

```
                              Final Results

                         Design                 8
                         Choice Sets          243
                         Alternatives           6
                         Parameters            35
                         Maximum Parameters  1215
                         D-Efficiency      7.1536
                         D-Error           0.1398
```

Using the choice design constructed directly from the full-factorial design, we get a *D*-efficiency of 6.5. With the search, we get 7.15. We really have no way of knowing the maximum *D*-efficiency for this problem. We cannot even use the standardized orthogonal contrast coding. For this type of design, with all of the interactions involved in the coding, there is no reason to believe that a choice design constructed from an orthogonal array is going to good. At 243 choice sets, this design is too large to use without breaking it up into many blocks. However, we can use the 7.15 efficiency value to scale efficiency for smaller designs to a scale from 0 to approximately 100. The following step illustrates:

```
%let sets = 32;

%choiceff(data=crosscan,           /* candidate set of choice sets         */
          model=class(b            /* brand effects                        */
                b*p / zero=' ')    /* alternative-specific effects         */
                class(b / zero=none)/* cross effects                       */
                * identity(x1-x5),
          maxiter=10,              /* maximum number of designs to make    */
          drop=B1X1 B2X2 B3X3      /* drop cross effects of brand on self  */
               B4X4 B5X5,
          rscale=&sets * 7.1536/243,/* scaling factor for relative eff     */
          nsets=&sets,             /* number of choice sets                */
          nalts=6,                 /* number of alternatives               */
          seed=104,                /* random number seed                   */
          beta=zero)               /* assumed beta vector, Ho: b=0         */
```

The raw *D*-efficiency and the relative *D*-efficiency (scaled to range from 0 to approximately 100) are as follows:

```
                              Final Results

                        Design                  3
                        Choice Sets            32
                        Alternatives            6
                        Parameters             35
                        Maximum Parameters    160
                        D-Efficiency       0.9390
                        Relative D-Eff    99.6825
                        D-Error            1.0649
                        1 / Choice Sets    0.0313
```

Relative to an unknown optimal design in 32 choice sets, this design is *approximately* 99.68% *D*-efficient. The *D*-efficiency is scaled relative to the number of choice sets times the proportion consisting of the approximate maximum *D*-efficiency divided by the number of choice sets in the comparison design. With 243 choice sets, *D*-efficiency is 7.1536. So 7.1536/243 provides the per set efficiency in the larger design. The number of sets is multiplied by this fraction to get the maximum expected *D*-efficiency for the smaller design. Since we do not know the maximum *D*-efficiency, you could be conservative and specify: `rscale=&sets * 7.2 / 243`, `rscale=&sets * 7.5 / 243`, or some other value.

Whenever you do not know the maximum *D*-efficiency, you can use this approach. Create a large design directly from a large candidate set or by searching a large candidate set. Then use its *D*-efficiency (or a slightly larger value to be conservative) and number of choice sets to scale the *D*-efficiency of a smaller and more realistic design.

%ChoicEff Macro Options

The following options can be used with the `%ChoicEff` macro:

	Option	Description
	`help`	(positional) "help" or "?" displays syntax summary
	`bestcov=`*SAS-data-set*	covariance matrix for the best design
	`bestout=`*SAS-data-set*	best design
	`beta=`*list*	true parameters
	`chunks=`*n*	number of observations to code at once
	`converge=`*n*	convergence criterion
	`cov=`*SAS-data-set*	all of the covariance matrices
	`data=`*SAS-data-set*	input choice candidate set
	`drop=`*variable-list*	variables to drop from the model
	`fixed=`*variable-list*	variable that flags fixed alternatives
∗	`flags=`*variable-list*|*n*	variables that flag the alternatives
	`init=`*SAS-data-set*	input initial design data set
	`initvars=`*variable-list*	initial variables

∗ - a new option or an option with new features in this release.

	Option	Description
	`intiter=`*n*	maximum number of internal iterations
	`iter=`*n*	maximum iterations (designs to create)
	`maxiter=`*n*	maximum iterations (designs to create)
	`model=`*model-specification*	`model` statement list of effects
	`morevars=`*variable-list*	more variables to add to the model
	`n=`*n*	number of observations
	`nalts=`*n*	number of alternatives
	`nsets=`*n*	number of choice sets desired
	`options=coded`	displays the coded candidate set
	`options=detail`	displays the details of the swaps
$*$	`options=nobeststar`	no asterisk when a better design is found
	`options=nocode`	skips the PROC TRANSREG coding stage
	`options=nodups`	prevents duplicate choice set creation
	`options=notests`	suppress the hypothesis tests
	`options=orthcan`	orthogonalizes the candidate set
$*$	`options=outputall`	outputs all designs to OUT= and COV=
	`options=relative`	displays final relative *D*-efficiency
$*$	`options=resrep`	same as `options=detail`
	`out=`*SAS-data-set*	all designs data set
$*$	`restrictions=`*macro-name*	restrictions macro
$*$	`resvars=`*variable-list*	variables for restrictions
$*$	`rscale=`*r*	relative efficiency scaling factor
$*$	`rscale=generic`	equivalent to `rscale=n` (*n* sets)
$*$	`rscale=alt`	for simple alternative-specific designs
$*$	`rscale=partial=`*p* `of` *q*	*p* of *q* alternatives or attributes vary
	`seed=`*n*	random number seed
	`submat=`*number-list*	submatrix for efficiency calculations
	`types=`*integer-list*	number of sets of each type
	`typevar=`*variable*	choice set types variable
	`weight=`*weight-variable*	optional weight variable

$*$ - a new option or an option with new features in this release.

Help Option

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%choiceff(help)
%choiceff(?)
```

Required Options

You must specify both the `model=` and `nsets=` options and either the `flags=` or `nalts=` options. You can omit `beta=` if you just want a listing of effects, however you must specify `beta=` to create a design. The rest of the options are optional.

model= *model-specification*
specifies a PROC TRANSREG `model` statement, which lists the attributes and describes how they are coded. There are many potential forms for the model specification and a number of options. See the SAS/STAT PROC TRANSREG documentation. PROC TRANSREG has a new option with version 9.2 of SAS that is often useful in this macro, namely the standardized orthogonal contrast coding requested by the `sta` or `standorth` option. For some designs, with this option and a specification of `options=relative`, you can get a relative *D*-efficiency in the 0 to 100 range. If you are running an earlier version of SAS and cannot use this option, your functionality is in no way limited, but you will not have a 0 to 100 scale for relative *D*-efficiency.

The following option specifies generic effects:

```
model=class(x1-x3),
```

The following option specifies brand and alternative-specific effects:

```
model=class(b)
      class(b*x1 b*x2 b*x3 / effects zero=' '),
```

The following option specifies brand, alternative-specific, and cross effects:

```
model=class(b b*p / zero=' ')
      identity(x1-x5) * class(b / zero=none),
```

See pages 808 through 946 for other examples of `model` syntax. Furthermore, all of the PROC TRANSREG and `%ChoicEff` macro examples from pages 327 through 610 show examples of model syntax for choice models.

nsets= *n*
specifies the number of choice sets desired.

Other Required Options

You must specify exactly one of these next two options. When the candidate set consists of individual alternatives to be swapped, specify the alternative flags with `flags=`. When the candidate set consists of entire sets of alternatives to be swapped, specify the number of alternatives in each set with `nalts=`.

flags= *variable-list* | *number-of-alternatives*
specifies variables that flag the alternatives for which each candidate can be used. There must be one flag variable per alternative. If every candidate can be used in all alternatives, then the flags are constant. When the flags are all constant (in a purely generic design), you can have the macro create these flag variables for you by specifying the number of alternatives rather than a list of flag variables. Example: `flags=3`. Alternatively, you can make the flag variables yourself. For example, with three alternatives, create these constant flags: `f1=1 f2=1 f3=1`.

Otherwise, with designs with brands or alternative labels, with three alternatives, specify `flags=f1-f3` and create a candidate set where: alternative 1 candidates are indicated by `f1=1 f2=0 f3=0`, alternative 2 candidates are indicated by `f1=0 f2=1 f3=0`, and alternative 3 candidates are indicated by `f1=0 f2=0 f3=1`.

nalts= *n*

specifies the number of alternatives in each choice set for the set-swapping algorithm.

Other Options

The rest of the parameters are optional. You can specify zero or more of them.

bestcov= *SAS-data-set*

specifies a name for the data set containing the covariance matrix for the best design. By default, this data set is called BESTCOV.

bestout= *SAS-data-set*

specifies a name for the data set containing the best design. By default, this data set is called BEST. Often, you will want to specify a two-level name to create a permanent SAS data set so the design is available later for analysis.

beta= *list*

specifies the true parameters. By default, when `beta=` is not specified, the macro just reports on coding. You can specify `beta=zero` to assume all zeros. Otherwise specify a number list: `beta=1 -1 2 -2 1 -1`.

chunks= *n*

specifies the number of observations to process at one time with the coding step and PROC TRANSREG. By default, the entire data set is processed at once. You can specify a value, say 1/2 or 1/3 of the number of choice sets times the number of alternatives to break up the coding into smaller chunks if you run out of memory. Ideally, make the value a multiple of the number of choice sets. Be sure that you do not leave one or a few extra observations in the last chunk, particularly if you are using one of the orthogonal codings (for example, `sta`) or you will get an error. Usually, you will not need to specify this option.

converge= *n*

specifies the *D*-efficiency convergence criterion. By default, `converge=0.005`.

cov= *SAS-data-set*

specifies a name for the data set containing all of the covariance matrices for all of the designs. By default, this data set is called COV.

data= *SAS-data-set*
specifies the input choice candidate set. By default, the macro uses the last data set created.

drop= *variable-list*
specifies a list of variables to drop from the model. If you specified a less-than-full-rank model in the `model=` specification, you can use `drop=` to produce a full rank coding. When there are redundant variables, the macro displays a list that you can use in the `drop=` option in a subsequent run.

fixed= *variable-list*
specifies the variable that flags the fixed alternatives. When `fixed=variable` is specified, the `init=` data set must contain the named variable, which indicates which alternatives are fixed (cannot be swapped out) and which ones can be changed. Example: `fixed=fixed, init=init, initvars=x1-x3`. Values of the `fixed=` variable include:

1 – means this alternative can never be swapped out.

0 – means this alternative is used in the initial design, but it can be swapped out.

. – means this alternative should be randomly initialized, and it can be swapped out.

The `fixed=` option can be specified only when both `init=` and `initvars=` are specified.

init= *SAS-data-set*
specifies an input initial design data set. Null means a random start. One usage is to specify the `bestout=` data set for an initial start. When `flags=` is specified, `init=` must contain the index variable. Example: `init=best(keep=index)`. When `nalts=` is specified, `init=` must contain the choice set variable. Example: `init=best(keep=set)`.

Alternatively, the `init=` data set can contain an arbitrary design, potentially created outside this macro. In that case, you must also specify `initvars=factors`, where factors are the factors in the design, for example, `initvars=x1-x3`. When alternatives are swapped, this data set must also contain the `flags=` variables. When `init=` is specified with `initvars=`, the data set can also contain a variable specified in the `fixed=` option, which indicates which alternatives are fixed, and which ones can be swapped in and out.

intiter= *n*
specifies the maximum number of internal iterations. Specify `intiter=0` to just evaluate efficiency of an existing design. By default, `intiter=10`.

initvars= *variable-list*
specifies the factor variables in the `init=` data set that must match up with the variables in the `data=` data set. See `init=`. All of these variables must be of the same type.

maxiter= *n*
iter= *n*
specifies the maximum iterations (designs to create). By default, `maxiter=10`.

morevars= *variable-list*
specifies more variables to add to the model. This option gives you the ability to specify a list of variables to copy along as is, through the TRANSREG coding, then add them to the model.

n= *n*
specifies the number of observations to use in the variance matrix formula. By default, `n=1`.

options= *options-list*
specifies binary options. By default, none of these options are specified. Specify one or more of the following values after `options=`.

`coded`
displays the coded candidate set.

`detail`
displays the details of the swaps and any restriction violations. This option adds more information to the iteration history tables than is displayed by default. You can use `options=resrep` as an alias for `options=detail`. The former is the name of the option in the `%MktEx` macro that provides a report on restriction violations and conformance. It is a good idea to specify this option with restrictions until you are sure that your restrictions macro is correct.

`nobeststar`
do not print an asterisk when a better design is found. By default, an asterisk is printed in the iteration history table whenever a design is found with a *D*-efficiency that is greater than the previous best.

`nocode`
skips the PROC TRANSREG coding stage, assuming that WORK.TMP_CAND was created by a previous step. This is most useful with set swapping when the candidate set can be big. It is important with `options=nocode` to note that the effect of `morevars=` and `drop=` in previous runs has already been taken care of, so do not specify them (unless for instance you want to drop still more variables).

`nodups`
prevents the same choice set from coming out more than once. This option does not affect the initialization, so the random initial design might have duplicates. This option forces duplicates out during the iterations, so do not set `intiter=` to a small value. It might take several iterations to eliminate all duplicates. It is possible that efficiency will decrease as duplicates are forced out. With set swapping, this macro checks the candidate choice set numbers to avoid duplicates. With alternative swapping, this macro checks the candidate alternative index to avoid duplicates. The macro does not look at the actual factors. This makes the checks faster, but if the candidate set contains duplicate choice sets or alternatives, the macro might not succeed in eliminating all duplicates. Run the `%MktDups` macro (which looks at the actual factors) on the design to check and make sure all duplicates are eliminated. If you are using set swapping to make a generic design make sure you run the `%MktDups` macro on the candidate set to eliminate duplicate choice sets in advance.

`notests`
suppresses displaying the diagonal of the covariance matrix, and hypothesis tests for this n and $\boldsymbol{\beta}$. When $\boldsymbol{\beta}$ is not zero, the results include a Wald test statistic (β divided by the standard error), which is normally distributed, and the probability of a larger squared Wald statistic.

`orthcan`
orthogonalizes the candidate set.

`outputall`
outputs all designs to the `out=` and `cov=` data sets. When the `maxiter=` value is less than or equal to 100, this option is the default. However, when the `maxiter=` value is greater than 100, only designs that improve on the previous best design are output by default. This is a change from previous releases.

`relative`
displays the relative D-efficiency for the final design, which is 100 times the D-efficiency divided by the number of choice sets. In other words, this option scales the D-efficiency relative to a (perhaps hypothetical) design with D-efficiency equal to the number of choice sets and displays it. When `beta=zero` is specified along with the standardized orthogonal contrast coding in the model specification and a generic choice design is requested, this scales D-efficiency to a 0 to 100 scale. Certain optimal generic choice designs constructed through combinatorial methods will have a relative D-efficiency of 100. While you can display this value for any other type of design and specification, it will not generally be on a 0 to 100 scale except in certain special cases, and this is why it is not displayed by default. You can specify the `rscale=` option if you have used the standardized orthogonal contrast coding and would like D-efficiency scaled relative to a value other than the number of choice sets. The following steps show an example of where it would make sense to specify `options=relative`:

`resrep`
is the same as `options=detail`.

```
%mktex(4 ** 5, n=16)

%mktlab(data=design, vars=Set x1-x4)

%choiceff(data=final,              /* candidate set of choice sets  */
          init=final(keep=set),     /* select these sets from cands  */
          intiter=0,                /* eval without internal iters   */
          model=class(x1-x4 / sta), /* model with stdz orthog coding */
          options=relative,         /* display relative D-efficiency */
          nsets=4,                  /* number of choice sets         */
          nalts=4,                  /* number of alternatives        */
          beta=zero)                /* assumed beta vector, Ho: b=0  */
```

The standardized orthogonal contrast coding is requested with the `sta` option in the `class` specification. If you are not running version 9.2 or a later SAS release, remove the slash and the `sta` option from the *model* specification. The final results table contains the relative *D*-efficiency in addition to all of the other usual results. In this case, since an optimal generic design is being evaluated, relative *D*-efficiency is 100.

out= *SAS-data-set*

specifies a name for the output SAS data set with all of the final designs. The default is `out=results`.

restrictions= *macro-name*

specifies the name of a restrictions macro, written in IML, that quantifies the badness of the design in an IML scalar that must be called `bad`. By default, there are no restrictions. When a restrictions macro is specified, then the `resvars=` option must be specified as well.

revars= *variable-list*

specifies the variables for restrictions. These variables must all be numeric. The resvars variables are available for your restrictions macro in the following matrices:

`xmat`	- the current design
`x`	- the choice set that is being considered for the `setnum` position (the current choice set number) in `xmat`

When restrictions are posed in terms of the entire design, the code in the restrictions macro might have the following form (where `^=` means not equals):

```
do s = 1 to nsets;
   if s ^= setnum then z = xmat[((s - 1) * nalts + 1) : (s * nalts),];
   else z = x;
   ... evaluate choice set z and accumulate badness ...
   end;
```

The index vector `((s - 1) * nalts + 1) :  (s * nalts)` (used as the row index in `xmat`) extracts one choice set. For example, with 3 alternatives, the index vector is: `1:3` when `s = 1` and extracts the first choice set from `xmat`, `4:6` when `s = 2` and extracts the second choice set from `xmat`, `7:9` when `s = 3`, and so on. The submatrix `xmat[((setnum - 1) * nalts + 1) :  (setnum * nalts),]` contains the `setnum` choice set that is currently in the design, and `x` contains the candidate choice set that is being evaluated. The submatrix `xmat[((setnum - 1) * nalts + 1) :  (setnum * nalts),]` is replaced by `x` when the replacement increases efficiency or decreases restriction violations.

The first `resvars=` variable is in the first column of `x` and `xmat` (`x[,1]` and `xmat[,1]`), ..., and the last `resvars=` variable is in the last column of `x` and `xmat` (`x[,m]` and `xmat[,m]`) where `m = ncol(xmat)`. The `resvars=` variables are typically the attributes, but they can contain additional information as well. All restrictions must be posed in terms of the values of `x` and `xmat` along with the following:

`nsets`	- number of choice sets in the design
`nalts`	- number of alternatives in the design
`setnum`	- number of current choice set
`altnum`	- number of current alternative (only available with alternative swapping)

rscale= *r* | `generic` | `alt` | `partial=`*p* `of` *q*
specifies the scaling factor to use for relative *D*-efficiency computations. When you specify `rscale=`, the option `options=relative` is implied. By default, when this option is not specified, the number of choice sets is used when `options=relative` is specified. If you specify `rscale=`*r*, where *r* is some number, then relative *D*-efficiency equals: 100 × *D*-efficiency / *r*. If you want relative *D*-efficiency, and you know that the number of choice sets is not the right scaling factor (perhaps because you have a constant alternative) and if you know the *D*-efficiency of an optimal design, you can specify it to get relative *D*-efficiency. Note that *r* must be a number and not an expression. However, you can use the `%sysevalf` function to evaluate an expression (for example, `rscale=%sysevalf(16 * 4 / 8)`).

The option `rscale=generic` (with *n* choice sets) is equivalent to `rscale=`*n*.

The option `rscale=alt` (with n choice sets, m alternatives, and p parameters, and $r = (n^{m-1}(n(m-1)/m^2)^{p-m+1})^{1/p})$ is equivalent to `rscale=`*r*. If a design has brand (or alternative label) effects such that brand i always occurs in alternative i, and all other effects are alternative-specific, then there are $m-1$ parameters with a maximum determinant of n, and the remaining $p-m+1$ have 1 of m alternatives contributing information to each set, and $m-1$ of m alternatives contribute information so the maximum *D*-efficiency is $n(m-1)/m^2$ for the $p-m+1$ parameters in that part of the design. This formula will not provide a true maximum for more complicated designs such as designs with constant alternatives.

The option `rscale=partial=`*p* `of` *q*, (where p and q are integers) is used with partial-profile designs (where p of q attributes vary) or with a generic choice design with a constant alternative (where p of q alternatives vary). It sets `rscale=` to np/q. For example, with 16 choice sets and 4 of 8 attributes varying, `rscale=partial=4 of 16` is equivalent to `rscale=%sysevalf(16 * 4 / 8)` and `rscale=8`.

seed= *n*
specifies the random number seed. By default, `seed=0`, and clock time is used as the random number seed. By specifying a random number seed, results should be reproducible within a SAS release for a particular operating system and for a particular version of the macro. However, due to machine and macro differences, some results might not be exactly reproducible everywhere, although you would expect the efficiency differences to be slight.

submat= *number-list*
specifies a submatrix for which efficiency calculations are desired. Specify an index vector. For example, with 3 three-level factors, `a`, `b`, and `c`, and the model `class(a b c a*b)`, specify `submat=1:6`, to see the efficiency of just the 6×6 matrix of main effects. Specify `submat=3:6`, to see the efficiency of just the 4×4 matrix of `b` and `c` main effects.

types= *integer-list*
specifies the number of sets of each type to put into the design. This option is used when you have multiple types of choice sets and you want the design to consist of only certain numbers of each type. This option can be specified with the set-swapping algorithm. The argument is an integer list. When you specify `types=`, you must also specify `typevar=`. Say you are creating a design with 30 choice sets, and you want the first 10 sets to consist of sets whose `typevar=` variable in the candidate set is type 1, and you want the rest to be type 2. You would specify `types=10 20`.

typevar= *variable*
specifies a variable in the candidate data set that contains choice set types. The types must be integers starting with 1. This option can only be specified with the set-swapping algorithm. When you specify `typevar=`, you must also specify `types=`.

weight= *weight-variable*
specifies an optional weight variable. Typical usage is with an availability design. Give unavailable alternatives a weight of zero and available alternatives a weight of one. The number of alternatives must always be constant, so varying numbers of alternatives are handled by giving unavailable or unseen alternatives a weight of zero.

%ChoicEff Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktAllo Macro

The `%MktAllo` autocall macro manipulates data for an allocation choice experiment. See the page 535 for an example. The `%MktAllo` macro takes as input a data set with one row for each alternative of each choice set. For example, in a study with 10 brands plus a constant alternative and 27 choice sets, there are $27 \times 11 = 297$ observations in the input data set. The following output displays an example of an input data set:

```
Obs    Set     Brand       Price    Count

  1     1                               0
  2     1     Brand  1      $50       103
  3     1     Brand  2      $75        58
  4     1     Brand  3      $50       318
  5     1     Brand  4     $100        99
  6     1     Brand  5     $100        54
  7     1     Brand  6     $100        83
  8     1     Brand  7      $75        71
  9     1     Brand  8      $75        58
 10     1     Brand  9      $75       100
 11     1     Brand 10      $50        56
  .
  .
  .
296    27     Brand  9     $100        94
297    27     Brand 10      $50        65
```

It contains a choice set variable, product attributes (`Brand` and `Price`) and a frequency variable (`Count`) that contains the total number of times that each alternative was chosen.

The end result is a data set with twice as many observations that contains the number of times each alternative was chosen and the number of times it was not chosen. This data set also contains a variable `c` with a value of 1 for first choice and 2 for second or subsequent choice. A portion of this data set is as follows:

```
Obs    Set     Brand       Price   Count     c

  1     1                              0     1
  2     1                           1000     2
  3     1     Brand  1      $50      103     1
  4     1     Brand  1      $50      897     2
  5     1     Brand  2      $75       58     1
  6     1     Brand  2      $75      942     2
  7     1     Brand  3      $50      318     1
  8     1     Brand  3      $50      682     2
```

```
                                 .
                                 .
                                 .
                               593     27     Brand 10     $50        65    1
                               594     27     Brand 10     $50       935    2
```

The following step shows how you use the `%MktAllo` macro:

```
%mktallo(data=allocs2, out=allocs3, nalts=11,
         vars=set brand price, freq=Count)
```

The option `data=` names the input data set, `out=` names the output data set, `nalts=` specifies the number of alternatives, `vars=` names the variables in the data set that are used in the analysis excluding the `freq=` variable, and `freq=` names the frequency variable.

%MktAllo Macro Options

The following options can be used with the `%MktAllo` macro:

Option	Description
`help`	(positional) “help” or “?” displays syntax summary
`data=`*SAS-data-set*	input SAS data set
`freq=`*variable*	frequency variable
`nalts=`*n*	number of alternatives
`out=`*SAS-data-set*	output SAS data set
`vars=`*variable-list*	input variables

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%mktallo(help)
%mktallo(?)
```

You must specify the `nalts=`, `freq=`, and `vars=` options.

data= *SAS-data-set*

specifies the input SAS data set. By default, the macro uses the last data set created.

freq= *variable*

specifies the frequency variable, which contains the number of times this alternative was chosen. This option must be specified.

nalts= *n*

specifies the number of alternatives (including if appropriate the constant alternative). This option must be specified.

out= *SAS-data-set*
specifies the output SAS data set. The default is `out=allocs`.

vars= *variable-list*
specifies the variables in the data set that are used in the analysis but not the `freq=` variable. This option must be specified.

%MktAllo Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktBal Macro

The `%MktBal` autocall macro creates factorial designs using an algorithm that ensures that the design is perfectly balanced, or when the number of levels of a factor does not divide the number of runs, as close to perfectly balanced as possible. Before using the `%MktBal` macro, you should try the `%MktEx` macro to see if it makes a design that is balanced enough for your needs. The `%MktEx` macro can directly create thousands of orthogonal and balanced designs that the `%MktBal` algorithm will never find. Even when the `%MktEx` macro cannot create an orthogonal and balanced design, it will usually find a nearly balanced design. Designs created with the `%MktBal` macro, while perfectly balanced, might be less efficient than designs found with the `%MktEx` macro, and for large problems, the `%MktBal` macro can be slow.

The `%MktBal` macro is *not* a full-featured experimental design generator. For example, you cannot specify interactions that you want to estimate or specify restrictions such as which levels may or may not appear together. You must use the `%MktEx` macro for that. The `%MktBal` macro builds a design by creating a balanced first factor, optimally (or nearly optimally) blocking it to create the second factor, then blocking the first two factors to create the third, and so on. Once it creates all factors, it refines each factor. Each factor is in turn removed from the design, and the rest of the design is reblocked, replacing the initial factor if the new design is more *D*-efficient.

The following steps provide a simple example of creating and evaluating a design with 2 two-level factors and 3 three-level factors in 18 runs:

```
%mktbal(2 2 3 3 3, n=18, seed=151)

%mkteval(data=design)
```

The `%MktEval` macro evaluates the results. This design is in fact optimal.

In all cases, the factors are named `x1`, `x2`, `x3`, and so on. You can use the `%MktLab` macro to conveniently change them.

This next example, at 120 runs and with factor levels greater than 5, is starting to get big, and by default, it will run slowly. You can use the `maxstarts=`, `maxtries=`, and `maxiter=` options to make the macro run more quickly. The following steps create the design, with and without these options:

```
%mktbal(2 3 4 5 6 7 8 9 10, n=120, options=progress, seed=17)

%mktbal(2 3 4 5 6 7 8 9 10, n=120, options=progress, seed=17,
        maxstarts=1, maxiter=1, maxtries=1)
```

The second example, with the options, runs much faster than the first.

%MktBal Macro Options

The following options can be used with the `%MktBal` macro:

Option	Description
`list`	(positional) list of the numbers of levels
	(positional) "help" or "?" displays syntax summary
`init=`*SAS-data-set*	initial input experimental design
`iter=`*n*	maximum iterations (designs to create)
`maxinititer=`*n*	maximum initialization iterations
`maxiter=`*n*	maximum iterations (designs to create)
`maxstarts=`*n*	maximum number of random starts
`maxtries=`*n*	times to try refining each factor
`n=`*n*	number of runs in the design
`options=noprint`	suppress the final *D*-efficiency
`options=nosort`	do not sort the final design
`options=oa`	an orthogonal array is sought
`options=progress`	reports on macro progress
`options=sequential`	factors are added but not refined
`out=`*SAS-data set*	output experimental design
`seed=`*n*	random number seed

Help Option

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%mktbal(help)
%mktbal(?)
```

list

specifies a list of the numbers of levels of all the factors. For example, for 3 two-level factors specify either `2 2 2` or `2 ** 3`. Lists of numbers, like `2 2 3 3 4 4` or a *levels**number of factors* syntax like: `2**2 3**2 4**2` can be used, or both can be combined: `2 2 3**4 5 6`. The specification `3**4` means 4 three-level factors. You must specify a list. Note that the factor list is a positional parameter. This means it must come first, and unlike all other parameters, it is not specified after a name and an equal sign.

n= *n*

specifies the number of runs in the design. You must specify `n=`. You can use the `%MktRuns` macro to get suggestions for values of `n=`.

out= *sas-data set*
specifies the output experimental design. The default is `out=design`.

These next options control some of the details of the `%MktBal` macro.

init= *sas-data set*
specifies the initial design. You can specify just the first few columns of the design, in this data set with column names `x1`, `x2`, ..., and these columns will never change. This is different from the `init=` data set in the `%MktEx` macro. It is often the case that the first few columns can be constructed combinatorially, and it is known that the optimal design will just add new columns to the initial few orthogonal columns. This option makes that process more efficient. By default, when `init=` is not specified, all columns are iteratively found and improved.

maxinititer= *n*
specifies the maximum number of random starts for each factor during the initialization stage. This is the number of iterations that PROC OPTEX performs for each factor during the initialization. With larger values, the macro tends to find slightly better designs at a cost of slower run times. The default is the value of `maxstarts=`.

maxiter= *n*
iter= *n*
specifies the maximum iterations (designs to create). By default, `maxiter=5`. This is the outer most set of iterations.

maxstarts= *n*
specifies the maximum number of random starts for each factor. This is the number of iterations that PROC OPTEX performs for each factor. With larger values, the macro tends to find slightly better designs at a cost of slower run times. The default is `maxstarts=10`.

maxtries= *n*
specifies the maximum number of times to try refining each factor after the initialization stage. The default is `maxtries=10`. Increasing the value of this option usually has little effect since the macro stops refining each design when the efficiency stabilizes.

options= *options-list*
specifies binary options. By default, none of these options are specified. Specify one or more of the following values after `options=`.

`noprint`
specifies that the final *D*-efficiency should not be displayed.

`nosort`
do not sort the final design.

`oa`
specifies that an orthogonal array is sought. Factors are added sequentially and they are acceptable only when D-efficiency reaches 100. With `options=oa`, you can use the `%MktBal` macro to search for small orthogonal arrays or in some cases augment existing orthogonal arrays.

`progress`
reports on the macro's progress. For large numbers of factors, a large number or runs, or when the number of levels is large, this macro is slow. The `options=progress` specification gives you information about which step is being executed.

`sequential`
sequentially adds factors to the design and does not go back and refine them.

seed= *n*
specifies the random number seed. By default, `seed=0`, and clock time is used to make the random number seed. By specifying a random number seed, results should be reproducible within a SAS release for a particular operating system and for a particular version of the macro. However, due to machine and macro differences, some results might not be exactly reproducible everywhere, although you would expect the efficiency differences to be slight.

%MktBal Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktBIBD Macro

The `%MktBIBD` autocall macro finds balanced incomplete block designs (BIBDs). A BIBD is a list of treatments that appear together in blocks. Each block contains a subset of the treatments. BIBDs can be used in marketing research to construct partial-profile designs (Chrzan and Elrod 1995). The entries in the BIBD say which attributes are to be shown in each set. For example, a BIBD could be used for the situation where there are t attributes or messages (treatments) and k are shown at a time. A total of b sets of attributes (blocks) are shown. For examples of using this macro in the partial-profile context, see page 1145. See the following pages for examples of using this macro in the discrete choice chapter: 642, 645, 650, 654, 656, 659 and 662. Additional examples appear throughout this chapter.

BIBDs are also used in marketing research to construct MaxDiff (best-worst) designs (Louviere 1991, Finn and Louviere 1992). In a MaxDiff study, subjects are shown sets (blocks) of messages or product attributes (treatments) and are asked to choose the best (or most important) from each set as well as the worst (or least important). For examples of using this macro in the MaxDiff context, see page 1105.

The documentation for the `%MktPPro` and `%MktMDiff` macros discuss the `%MktBIBD` macro in the context of sets and attributes, whereas most of the `%MktBIBD` macro documentation uses the more traditional statistical vocabulary of blocks and treatments. Furthermore, the output of the `%MktBIBD` macro can correspond to the marketing research vocabulary (when the **nattrs=** option is specified) or the statistical vocabulary (when the **t=** option is specified). The **t=** and **nattrs=** options are otherwise aliases for each other and specify the number of treatments (or attributes).

The following design is an example of a BIBD:

```
                    Balanced Incomplete Block Design

                              x1    x2    x3

                               3     5     2
                               1     5     2
                               3     2     4
                               4     3     1
                               5     3     1
                               1     2     3
                               4     1     5
                               5     4     3
                               2     1     4
                               2     4     5
```

This BIBD has $b = 10$ blocks of treatments. Each of the 10 rows is a block. There are $t = 5$ treatments, since the entries in the design are the integers 1 through 5. Each block consists of $k = 3$ treatments. Note that the number of entries in the design, $b \times k = 10 \times 3 = 6 \times t = 30$. Each of the $t = 5$ treatments occurs exactly 6 times in the design, and each treatment occurs with every other treatment 3 times. This can be seen in the following treatment by treatment frequencies:

```
                    Treatment by Treatment Frequencies

                              1  2  3  4  5

                           1  6  3  3  3  3
                           2     6  3  3  3
                           3        6  3  3
                           4           6  3
                           5              6
```

When the %MktBIBD macro makes a BIBD, it tries to optimize the treatment by position frequencies. In other words, it tries to ensure that each treatment occurs in each of the *k* positions equally often, or at least close to equally often. The following are the treatment by position frequencies for this design, which in this case are perfect:

```
                    Treatment by Position Frequencies

                                 1  2  3

                              1  2  2  2
                              2  2  2  2
                              3  2  2  2
                              4  2  2  2
                              5  2  2  2
```

The following %MktBIBD macro step generates this BIBD:

```
   %mktbibd(b=10, t=5, k=3, seed=104)
```

In addition to the three tables above, the %MktBIBD macro also displays the following summary information:

```
            Block Design Efficiency Criterion       100.0000
            Number of Treatments, t                        5
            Block Size, k                                  3
            Number of Blocks, b                           10
            Treatment Frequency                            6
            Pairwise Frequency                             3
            Total Sample Size                             30
            Positional Frequencies Optimized?            Yes
```

The fact the that efficiency is 100 shows that %MktBIBD found a BIBD. In many cases, it will find a design that is close to a BIBD but each of the pairwise frequencies is not constant. For many marketing research problems, this is good enough.

The BIBD is available in the `out=BIBD` data set. The following step displays the design:

```
   proc print data=bibd noobs; run;
```

The results are as follows:

```
                              x1    x2    x3

                               3     5     2
                               1     5     2
                               3     2     4
                               4     3     1
                               5     3     1
                               1     2     3
                               4     1     5
                               5     4     3
                               2     1     4
                               2     4     5
```

Every BIBD has a binary representation, or *incidence matrix*, with *b* rows and *t* columns and *k* ones that indicate which treatments appear in each block. The following step displays the `outi=incidence` matrix:

```
   proc print data=incidence noobs; run;
```

The results are as follows:

```
                          b1    b2    b3    b4    b5

                           0     1     1     0     1
                           1     1     0     0     1
                           0     1     1     1     0
                           1     0     1     1     0
                           1     0     1     0     1
                           1     1     1     0     0
                           1     0     0     1     1
                           0     0     1     1     1
                           1     1     0     1     0
                           0     1     0     1     1
```

This shows that the first block consists of treatments 2, 3, and 5, just as the `out=BIBD` design does, but in a different format.

You can also view the BIBD arrayed as a block by treatment factorial design. The following step displays the first 9 (out of 30) observations:

```
proc print data=factorial(obs=9) noobs; run;
```

The results are as follows:

```
                              Block    Treatment

                                1          3
                                1          5
                                1          2
                                2          1
                                2          5
                                2          2
                                3          3
                                3          2
                                3          4
```

The following %MktBIBD macro step generates a BIBD with nondirectional row-neighbor balance:

```
%mktbibd(b=14, t=7, k=4, options=neighbor, seed=104)
```

The results are as follows:

```
                 Block Design Efficiency Criterion       100.0000
                 Number of Treatments, t                        7
                 Block Size, k                                  4
                 Number of Blocks, b                           14
                 Treatment Frequency                            8
                 Pairwise Frequency                             4
                 Total Sample Size                             56
                 Positional Frequencies Optimized?            Yes
                 Row-Neighbor Frequencies Optimized?          Yes

                        Treatment by Treatment Frequencies

                               1  2  3  4  5  6  7

                            1  8  4  4  4  4  4  4
                            2     8  4  4  4  4  4
                            3        8  4  4  4  4
                            4           8  4  4  4
                            5              8  4  4
                            6                 8  4
                            7                    8
```

```
                    Treatment by Position Frequencies

                                1  2  3  4

                             1  2  2  2  2
                             2  2  2  2  2
                             3  2  2  2  2
                             4  2  2  2  2
                             5  2  2  2  2
                             6  2  2  2  2
                             7  2  2  2  2

                       Row-Neighbor Frequencies

                           1  2  3  4  5  6  7

                        1     2  2  2  2  2  2
                        2        2  2  2  2  2
                        3           2  2  2  2
                        4              2  2  2
                        5                 2  2
                        6                    2
                        7

                   Balanced Incomplete Block Design

                          x1    x2    x3    x4

                           3     4     2     5
                           5     6     1     4
                           1     4     5     3
                           7     1     6     4
                           4     5     7     2
                           5     3     7     6
                           3     7     5     1
                           1     2     4     7
                           6     2     1     5
                           4     7     6     3
                           2     3     4     6
                           2     6     3     1
                           6     5     2     7
                           7     1     3     2
```

The resulting design is a BIBD with each of the 7 treatments appearing in each of the 4 positions within a block exactly twice. Furthermore, the row-neighbor frequencies show that each of the $7 \times (7-1)/2 = 21$ pairs of treatments occur exactly twice. The pairs in this design are 3 with 4, 4 with 2, 2 with 5, 5 with 6, and so on. The order of the treatments in each pair is ignored. Hence, in this design, with the nondirectional row-neighbor balance, the 2 followed by 5 in the first row of the design is treated the same as the 5 followed by 2 in the second to last row of the design. This design is usually easily found in a few seconds. The last line of the first table ("Row-Neighbor Frequencies Optimized? Yes") along with the constant row-neighbor frequencies, shows that perfect row-neighbor balance was achieved.

The following finds a BIBD with row-neighbor balance where the order of the treatments *does* matter using `options=serial`:

```
%mktbibd(b=14, t=7, k=4, options=serial, seed=361699)
```

The results are as follows:

```
Block Design Efficiency Criterion       100.0000
Number of Treatments, t                        7
Block Size, k                                  4
Number of Blocks, b                           14
Treatment Frequency                            8
Pairwise Frequency                             4
Total Sample Size                             56
Positional Frequencies Optimized?            Yes
Row-Neighbor Frequencies Optimized?          Yes

       Treatment by Treatment Frequencies

                1  2  3  4  5  6  7

             1  8  4  4  4  4  4  4
             2     8  4  4  4  4  4
             3        8  4  4  4  4
             4           8  4  4  4
             5              8  4  4
             6                 8  4
             7                    8

       Treatment by Position Frequencies

                    1  2  3  4

                 1  2  2  2  2
                 2  2  2  2  2
                 3  2  2  2  2
                 4  2  2  2  2
                 5  2  2  2  2
                 6  2  2  2  2
                 7  2  2  2  2
```

```
              Row-Neighbor Frequencies

                 1  2  3  4  5  6  7

              1     1  1  1  1  1  1
              2  1     1  1  1  1  1
              3  1  1     1  1  1  1
              4  1  1  1     1  1  1
              5  1  1  1  1     1  1
              6  1  1  1  1  1     1
              7  1  1  1  1  1  1

          Balanced Incomplete Block Design

                x1    x2    x3    x4

                 1     4     2     5
                 3     1     5     7
                 4     5     1     7
                 2     3     7     1
                 6     7     5     3
                 7     6     4     1
                 7     2     6     5
                 1     3     4     6
                 3     5     2     4
                 4     7     3     2
                 5     4     3     6
                 5     6     1     2
                 6     2     7     4
                 2     1     6     3
```

In this set of output, with `options=serial`, the row-neighbor frequencies appear both above and below the diagonal. In contrast, in the previous set of output, with `options=neighbor`, the row-neighbor frequencies only appeared above the diagonal. You can see that each pair occurs exactly once in this design. In this design, with the serial (directional) row-neighbor balance, the 2 followed by 5 in the first row of the design is *not* treated the same as the 5 followed by 2 in the ninth row of the design.

An `options=serial` design is typically much harder to find than an `options=neighbor` design. In this case, a random number seed that is known to produce an optimal design reasonably quickly was chosen.* Usually, you will have to run the macro more than once or change some options such as the time value in the `positer=` option and iterate for up to a few hours to find an equivalent design.

The following requests this same design with a different seed:

```
%mktbibd(b=14, t=7, k=4, options=serial, seed=104)
```

*By specifying a random number seed, results should be reproducible within a SAS release for a particular operating system and for a particular version of the macro. However, due to machine and macro differences, results might not be exactly reproducible everywhere.

A portion of the output is as follows:

```
                    Positional Frequencies Optimized?            No
                    Row-Neighbor Frequencies Optimized?          No

                           Treatment by Position Frequencies

                                        1  2  3  4

                                     1  2  2  2  2
                                     2  2  3  2  1
                                     3  2  2  2  2
                                     4  2  2  2  2
                                     5  2  2  2  2
                                     6  2  2  2  2
                                     7  2  1  2  3

                                Row-Neighbor Frequencies

                                    1  2  3  4  5  6  7

                                 1     1  1  1  1  1  1
                                 2  1     1  1  1  2  1
                                 3  1  1     1  1  1  1
                                 4  1  1  1     1  1  1
                                 5  1  1  1  1     1  1
                                 6  1  1  1  1  1     1
                                 7  1  1  1  1  1  0
```

With most seeds and the default amount of iteration you will get results like these with nearly optimal position and row neighbor frequencies.

The following requests a design where it is not possible to have constant frequencies in the row-neighbor frequencies matrix:

```
   %mktbibd(b=7, t=7, k=4, options=serial, seed=104)
```

A portion of the output is as follows:

```
                    Positional Frequencies Optimized?           Yes
                    Row-Neighbor Frequencies Optimized?         Yes
```

```
                        Row-Neighbor Frequencies

                           1  2  3  4  5  6  7

                        1     0  1  1  1  0  0
                        2  0     1  0  0  1  1
                        3  1  0     0  0  1  1
                        4  0  1  0     1  1  0
                        5  1  1  0  0     0  1
                        6  0  0  1  1  1     0
                        7  1  1  0  1  0  0
```

The `%MktBIBD` macro reports that the row-neighbor frequencies are optimized since a mix of zeros and ones with no twos or larger values is optimal for this specification. However, the nonconstant frequencies show that row-neighbor balance is not possible for this BIBD.

BIBD Parameters

The parameters of a BIBD are:

- b specifies the number of blocks. In a partial-profile design, this is the number of profiles. In a MaxDiff design, this is the number of sets.
- t specifies the number of treatments. In a partial-profile or MaxDiff design, this is the total number of attributes or messages.
- k specifies the block size, which is the number of treatments in each block. In a partial-profile or MaxDiff design, this is the number of attributes or messages shown at one time.

When $r = b \times k/t$ and $l = r \times (k-1)/(t-1)$ are integers, and $k = t$ and $b \geq t$, then a complete block design might be possible. This is a necessary but not sufficient condition for the existence of a complete block design. When $r = b \times k/t$ and $l = r \times (k-1)/(t-1)$ are integers, and $k < t$ and $b \geq t$, then a balanced incomplete block design might be possible. This is a necessary but not sufficient condition for the existence of a BIBD. You can use the macro `%MktBSize` to find parameters in which BIBDs might exist. The `%MktBIBD` macro uses PROC OPTEX and a computerized search to find BIBDs. It does not have a library of BIBDs or use combinatorial constructions. Hence, it will not always find a BIBD even when one is known to exist. However, it usually works quite well in finding small BIBDs or something close for larger problems. The following step displays some of the smaller specifications for which a BIBD might exist:

```
%mktbsize(t=1 to 20, k=2 to 0.5 * t, b=t to 100)
```

The results are as follows:

```
                                                                        n
         t            k          b             r          Lambda      Total
     Number of      Block     Number       Treatment     Pairwise      Sample
     Treatments     Size      of Blocks    Frequency    Frequencies    Size

          4           2          6             3             1           12
          5           2         10             4             1           20
          6           2         15             5             1           30
          6           3         10             5             2           30
          7           2         21             6             1           42
          7           3          7             3             1           21
          8           2         28             7             1           56
          8           3         56            21             6          168
          8           4         14             7             3           56
          9           2         36             8             1           72
          9           3         12             4             1           36
          9           4         18             8             3           72
         10           2         45             9             1           90
         10           3         30             9             2           90
         10           4         15             6             2           60
         10           5         18             9             4           90
         11           2         55            10             1          110
         11           3         55            15             3          165
         11           4         55            20             6          220
         11           5         11             5             2           55
         12           2         66            11             1          132
         12           3         44            11             2          132
         12           4         33            11             3          132
         12           6         22            11             5          132
         13           2         78            12             1          156
         13           3         26             6             1           78
         13           4         13             4             1           52
         13           5         39            15             5          195
         13           6         26            12             5          156
         14           2         91            13             1          182
         14           4         91            26             6          364
         14           6         91            39            15          546
         14           7         26            13             6          182
         15           3         35             7             1          105
         15           5         21             7             2          105
         15           6         35            14             5          210
         15           7         15             7             3          105
```

16	3	80	15	2	240
16	4	20	5	1	80
16	5	48	15	4	240
16	6	16	6	2	96
16	7	80	35	14	560
16	8	30	15	7	240
17	4	68	16	3	272
17	5	68	20	5	340
17	8	34	16	7	272
18	6	51	17	5	306
18	9	34	17	8	306
19	3	57	9	1	171
19	4	57	12	2	228
19	6	57	18	5	342
19	7	57	21	7	399
19	9	19	9	4	171
20	4	95	19	3	380
20	5	76	19	4	380
20	8	95	38	14	760
20	10	38	19	9	380

%MktBIBD Macro Options

The following options can be used with the `%MktBIBD` macro:

Option	Description
`help`	(positional) “help” or “?” displays syntax summary
`b=`*b*	number of blocks (alias for `nsets=`)
`group=`*n*	number of groups
`k=`*k*	block size (alias for `setsize=`)
`nattrs=`*t*	number of attributes (alias for `t=`)
`nsets=`*b*	number of sets (alias for `b=`)
`optiter=`*n1* < *n2* < *n3* >>	number of PROC OPTEX iterations
`options=position`	optimizes position frequencies
`options=neighbor`	nondirectional row-neighbor balance and position
`options=serial`	directional row-neighbor balance and position
`out=`*SAS-data-set*	output data set BIBD
`outf=`*SAS-data-set*	output data set factorial design
`outi=`*SAS-data-set*	output data set incidence matrix
`outs=`*SAS-data-set*	output data set sorted design
`positer=`*n1* < *n2* < *n3* >>	number of iterations position frequencies
`seed=`*n*	random number seed
`setsize=`*k*	set size (alias for `k=`)
`t=`*t*	number of treatments (alias for `nattrs=`)
`weights=`*n n*	position frequency badness weights

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%mktbibd(help)
%mktbibd(?)
```

b= *b*
nsets= *b*

specifies the number of blocks. In a partial-profile design, this is the number of profiles. In a MaxDiff design, this is the number of sets. The `nsets=` and `b=` options are aliases. This option (in one of its two forms) must be specified.

group= *n*

specifies the number of groups into which the design is to be divided. This could be useful with MaxDiff and partial profiles. By default, the design is not divided into groups.

k= *k*
setsize= *k*

specifies the block size, which is the number of treatments in each block. In a partial-profile or MaxDiff design, this is the number of attributes or messages shown at one time in each set. The `setsize=` and `k=` options are aliases. This option (in one of its two forms) must be specified.

optiter= *n1 < n2 < n3 >>*

specifies the number of PROC OPTEX iterations. If one number is specified, it is just the number of iterations. With two numbers, *n1* and *n2*, *n1* iterations are performed and then efficiency is checked. If the block design efficiency criterion is 100, the iterations stop. Otherwise, this process is repeated up to *n2* times for a maximum of $n1 \times n2$ iterations. The default for *n2*, when it is missing, is 1000 when the parameters conform to the necessary conditions for a BIBD and 5 otherwise. The third value is the maximum amount of time in minutes to spend in PROC OPTEX. The default for the time value, when it is missing, is 5 minutes when the parameters conform to the necessary conditions for a BIBD and 0.5 minutes otherwise. Hence, when the parameters conform to the necessary conditions for a BIBD, the default is `optiter=500 1000 5`, otherwise, the default is `optiter=500 5 0.5`. For larger problems, you might want to specify values smaller than the defaults.

options= *options-list*

specifies binary options. By default, `options=position`.

`position`
optimizes position frequencies. The goal is for each treatment to appear equally often in each position.

`neighbor`
optimizes nondirectional row-neighbor balance and also position. The goal is for pairs of treatments, constructed from each of the first $k-1$ treatments in each block along with the treatment that follows it, to occur equally often in the design. The order of the treatments within each pair does not matter in evaluating row-neighbor balance. That is, treatment 1 appearing before treatment 2 is counted as the same as 2 appearing before 1.

`serial`
optimizes directional row-neighbor balance and also position. The goal is for pairs of treatments, constructed from each of the first $k-1$ treatments in each block along with the treatment that follows it, to occur equally often in the design. In contrast to `options=neighbor`, treatment 2 appearing before treatment 1 is treated as different from 1 appearing before 2.

out= *SAS-data-set*

specifies the output data set name for the $b \times k$ BIBD (or more generally, the incomplete block design). The default is `out=BIBD`.

outf= *SAS-data-set*

specifies the output data set name for the $bk \times 2$ factorial design matrix. The default is `outf=Factorial`.

outi= *SAS-data-set*

specifies the output data set name for the $b \times t$ incidence matrix. The default is `outi=Incidence`.

outs= *SAS-data-set*

specifies the output data set name for the $b \times k$ sorted design. The default is `outs=Sorted`.

positer= *n1 < n2 < n3 >>*

specifies the number of iterations for the algorithm that attempts to optimize the treatment by position frequencies. The default is `positer=200 5000 2`. The first value specifies the number of times to try to refine the design. The second value specifies the number of times to start from scratch with a different random start. Larger values increase the chances of finding better treatment by position frequencies at a cost of slower run times. The third value is the maximum amount of time in minutes to spend optimizing the positional frequencies. Specify `positer=` or `positer=0` to just request a design without optimizing position.

seed= *n*

specifies the random number seed. By default, `seed=0`, and clock time is used to make the random number seed. By specifying a random number seed, results should be reproducible within a SAS release for a particular operating system and for a particular version of the macro. However, due to machine and macro differences, some results might not be exactly reproducible everywhere, although you would expect the efficiency differences to be slight.

t= *t*
nattrs= *t*
specifies the number of treatments. In a partial-profile or MaxDiff design, this is the total number of attributes or messages. The `nattrs=` and `t=` options are aliases. This option (in one of its two forms) must be specified. When the `nattrs=` option is specified, the output will use the word "Attribute" rather than "Treatment" and "Set" rather than "Block".

weights= *n n*
specifies weights for position balance and row-neighbor balance. Specify two nonnegative numeric values. The total badness is a weighted sum of the position badness and the row-neighbor badness. The default is `weights=1 2`, so by default, row-neighbor balance is given the most weight. You can specify a weight of zero, for example, for the position balance (the first value) to just optimize row-neighbor balance.

Evaluating an Existing Block Design

The `%MktBIBD` macro does not have an option to evaluate existing block designs. However, you can run the following ad hoc macro to if you want to see the blocking design efficiency criterion, the treatment by position frequencies, and the treatment by treatment frequencies:

```
%macro mktbeval(design=_last_,      /* block design to evaluate         */
                attrortr=Attribute);/* string to print in the output    */
                                    /* specify Treatment or use default */

proc iml;
   use &design; read all into x;
   t = max(x);    k = ncol(x);    blocks = nrow(x);
   call symput('k', char(k));
   call symput('b', char(blocks));
   f = j(t, t, .);    p = j(t, k, 0);
   do i = 1 to blocks;
      do j = 1 to k;
         p[x[i,j],j] = p[x[i,j],j] + 1;
         do q = j to k;
            a = min(x[i,j],x[i,q]);
            b = max(x[i,j],x[i,q]);
            f[a,b] = sum(f[a,b], 1);
            end;
         end;
      end;
      options missing=' ';
      w = ceil(log10(t + 1)); t = right(char(1, w, 0) : char(t, w, 0));
      w = ceil(log10(k + 1)); q = right(char(1, w, 0) : char(k, w, 0));
      if max(f) < 100 then print "&attrortr by &attrortr Frequencies",,
         f[format=2. label='' rowname=t colname=t];
      else print "&attrortr by &attrortr Frequencies",,
         f[label='' rowname=t colname=t];
      if max(p) < 100 then print "&attrortr by Position Frequencies",,
         p[format=2. label='' rowname=t colname=q];
      else print "&attrortr by Position Frequencies",,
         p[label='' rowname=t colname=q];
      options missing='.';
      x = (1:blocks)` @ j(k, 1, 1) || shape(x, blocks * k);
      create __tmpbefac from x; append from x;
      quit;

   proc optex;
      class col2;
      model col2;
      generate initdesign=__tmpbefac method=sequential;
      blocks structure=(&b)&k init=chain iter=0;
      ods select BlockDesignEfficiencies;
      run;

   proc datasets nolist; delete __tmpbefac; run; quit;
   %mend;
```

The macro has two options. Specify the name of the block design in the `design=` option. By default, the treatments are labeled as "Attributes", however, you can specify `attrortr=Treatment` if you would like them labeled as treatments.

The following steps create a BIBD and evaluate it:

```
   %mktbibd(b=10, t=6, k=3, seed=104)

   %mktbeval;
```

The results of the evaluation step are as follows:

```
                    Attribute by Attribute Frequencies

                               1  2  3  4  5  6

                            1  5  2  2  2  2  2
                            2     5  2  2  2  2
                            3        5  2  2  2
                            4           5  2  2
                            5              5  2
                            6                 5
```

```
                      Attribute by Position Frequencies

                                   1  2  3

                                1  2  2  1
                                2  2  1  2
                                3  2  2  1
                                4  1  2  2
                                5  1  2  2
                                6  2  1  2

                           The OPTEX Procedure

          Design      Treatment        Treatment       Block Design
          Number     D-Efficiency     A-Efficiency     D-Efficiency
          ---------------------------------------------------------
               1        80.0000          80.0000         100.0000
```

These results match the results from the `%MktBIBD` macro (not shown). The efficiency result of interest is the block design *D*-efficiency, with is 100 for BIBDs.

%MktBIBD Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktBlock Macro

The `%MktBlock` autocall macro blocks a choice design or an ordinary linear experimental design. See the following pages for examples of using this macro in the discrete choice chapter: 426 and 497. Additional examples appear throughout this chapter. When a choice design is too large to show all choice sets to each subject, the design is blocked and a block of choice sets is shown to each subject. For example, if there are 36 choice sets, instead of showing each subject 36 sets, you could instead create 2 blocks and show 2 groups of subjects 18 sets each. You could also create 3 blocks of 12 choice sets or 4 blocks of 9 choice sets. You can also request just one block if you want to see the correlations and frequencies among all of the attributes of all of the alternatives of a choice design.

The design can be in one of two formats. Typically, a choice design has one row for each alternative of each choice set and one column for each of the attributes. Typically, this kind of design is produced by either the `%ChoicEff` or `%MktRoll` macro. Alternatively, a linear arrangement is an intermediate step in preparing some choice designs.* The linear arrangement has one row for each choice set and one column for each attribute of each alternative. Typically, the linear arrangement is produced by the `%MktEx` macro. The output from the `%MktBlock` macro is a data set containing the design, with the blocking variable added and hence not in the original order, with runs or choice sets nested within blocks.

The macro tries to create a blocking factor that is uncorrelated with every attribute of every alternative. In other words, the macro is trying to optimally add one additional factor, a blocking factor, to the linear arrangement. It is trying to make a factor that is orthogonal to all of the attributes of all of the alternatives. For linear arrangements, you can usually ask for a blocking factor directly as just another factor in the design, and then use the `%MktLab` macro to provide a name like `Block`, or you can use the `%MktBlock` macro.

The following steps create the blocking variable directly:

```
   %mktex(3 ** 7, n=27, seed=350)

   %mktlab(data=randomized, vars=x1-x6 Block)
```

The following steps create a design and then block it:

```
   %mktex(3 ** 6, n=27, seed=350)

   %mktblock(data=randomized, nblocks=3, seed=377, maxiter=50)
```

The results are as follows:

*See page 67 for an explanation of the linear arrangement of a choice design versus the arrangement of a choice design that is more suitable for analysis.

```
                Canonical Correlations Between the Factors
            There are 0 Canonical Correlations Greater Than 0.316

              Block    x1      x2      x3      x4      x5      x6

     Block    1        0       0       0       0       0       0
     x1       0        1       0       0       0       0       0
     x2       0        0       1       0       0       0       0
     x3       0        0       0       1       0       0       0
     x4       0        0       0       0       1       0       0
     x5       0        0       0       0       0       1       0
     x6       0        0       0       0       0       0       1

                          Summary of Frequencies
            There are 0 Canonical Correlations Greater Than 0.316

                            Frequencies

              Block       9 9 9
              x1          9 9 9
              x2          9 9 9
              x3          9 9 9
              x4          9 9 9
              x5          9 9 9
              x6          9 9 9
              Block x1    3 3 3 3 3 3 3 3 3
              Block x2    3 3 3 3 3 3 3 3 3
              Block x3    3 3 3 3 3 3 3 3 3
              Block x4    3 3 3 3 3 3 3 3 3
              Block x5    3 3 3 3 3 3 3 3 3
              Block x6    3 3 3 3 3 3 3 3 3
              x1 x2       3 3 3 3 3 3 3 3 3
              x1 x3       3 3 3 3 3 3 3 3 3
              x1 x4       3 3 3 3 3 3 3 3 3
              x1 x5       3 3 3 3 3 3 3 3 3
              x1 x6       3 3 3 3 3 3 3 3 3
              x2 x3       3 3 3 3 3 3 3 3 3
              x2 x4       3 3 3 3 3 3 3 3 3
              x2 x5       3 3 3 3 3 3 3 3 3
              x2 x6       3 3 3 3 3 3 3 3 3
```

```
x3 x4       3 3 3 3 3 3 3 3 3
x3 x5       3 3 3 3 3 3 3 3 3
x3 x6       3 3 3 3 3 3 3 3 3
x4 x5       3 3 3 3 3 3 3 3 3
x4 x6       3 3 3 3 3 3 3 3 3
x5 x6       3 3 3 3 3 3 3 3 3
N-Way       1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
            1 1 1 1 1 1 1 1
```

The output shows that the blocking factor is uncorrelated with all of the factors in the design. This output comes from the `%MktEval` macro, which is called by the `%MktBlock` macro.

The following output displays the blocked linear arrangement (3 blocks of nine choice sets):

```
Block    Run    x1    x2    x3    x4    x5    x6

  1       1      2     1     2     2     1     1
          2      3     2     1     1     3     2
          3      1     3     3     3     2     3
          4      2     3     1     3     1     2
          5      1     1     1     2     2     2
          6      2     2     3     1     1     3
          7      3     1     3     2     3     3
          8      3     3     2     3     3     1
          9      1     2     2     1     2     1

Block    Run    x1    x2    x3    x4    x5    x6

  2       1      1     3     3     1     3     2
          2      3     3     2     1     1     3
          3      2     2     3     2     2     2
          4      3     2     1     2     1     1
          5      2     1     2     3     2     3
          6      3     1     3     3     1     2
          7      2     3     1     1     2     1
          8      1     2     2     2     3     3
          9      1     1     1     3     3     1
```

```
                  Block    Run    x1    x2    x3    x4    x5    x6

                    3        1     3     2     1     3     2     3
                             2     3     1     3     1     2     1
                             3     1     3     3     2     1     1
                             4     2     2     3     3     3     1
                             5     2     1     2     1     3     2
                             6     1     2     2     3     1     2
                             7     3     3     2     2     2     2
                             8     2     3     1     2     3     3
                             9     1     1     1     1     1     3
```

Note that in the linear version of the design, there is one row for each choice set and all of the attributes of all of the alternatives are in the same row.

Next, we will create and block a choice design with two blocks of nine sets instead of blocking the linear version of a choice design. The following steps create and then block a choice design:

```
%mktex(3 ** 6, n=3**6)

%mktroll(design=design, key=2 3, out=out)

%choiceff(data=out,                 /* candidate set of choice sets         */
          model=class(x1-x3 / sta), /* model with stdz orthogonal coding    */
          nsets=18,                 /* number of choice sets                */
          nalts=2,                  /* number of alternatives               */
          seed=151,                 /* random number seed                   */
          options=nodups            /* do not create duplicate choice sets  */
                  relative,         /* display relative D-efficiency        */
          beta=zero)                /* assumed beta vector, Ho: b=0         */

* Block the choice design.  Ask for 2 blocks;
%mktblock(data=best, nalts=2, nblocks=2, factors=x1-x3, seed=472)
```

(Note that if this had been a branded example, and if you are running SAS version 8.2 or an earlier release, specify `id=brand`; do not add your brand variable to the factor list. For SAS 9.0 and later SAS releases, it is fine to add your brand variable to the factor list.)

Both the design and the blocking are not as good this time. The variable names in the output are composed of `Alt`, the alternative number, and the factor name. Since there are two alternatives each composed of three factors plus one blocking variable ($2 \times 3 + 1 = 7$), a 7×7 correlation matrix is reported. Some of the results are as follows:

```
                    Canonical Correlations Between the Factors
              There are 7 Canonical Correlations Greater Than 0.316

              Block    Alt1_x1   Alt1_x2   Alt1_x3   Alt2_x1   Alt2_x2   Alt2_x3

    Block     1         0.15      0         0.14      0.13      0.15      0.14
    Alt1_x1   0.15      1         0.41      0.21      0.51      0.20      0.30
    Alt1_x2   0         0.41      1         0.40      0.26      0.56      0.33
    Alt1_x3   0.14      0.21      0.40      1         0.19      0.31      0.52
    Alt2_x1   0.13      0.51      0.26      0.19      1         0.31      0.30
    Alt2_x2   0.15      0.20      0.56      0.31      0.31      1         0.48
    Alt2_x3   0.14      0.30      0.33      0.52      0.30      0.48      1

                             Summary of Frequencies
              There are 7 Canonical Correlations Greater Than 0.316
                        * - Indicates Unequal Frequencies

                                  Frequencies

                 Block            9 9
             *   Alt1_x1          8 5 5
             *   Alt1_x2          4 6 8
             *   Alt1_x3          6 7 5
             *   Alt2_x1          4 7 7
             *   Alt2_x2          8 5 5
             *   Alt2_x3          6 5 7

             *   Block Alt1_x1    4 2 3 4 3 2
             *   Block Alt1_x2    2 3 4 2 3 4
             *   Block Alt1_x3    3 3 3 3 4 2
             *   Block Alt2_x1    2 4 3 2 3 4
             *   Block Alt2_x2    4 3 2 4 2 3
             *   Block Alt2_x3    3 3 3 3 2 4

             *   Alt1_x1 Alt1_x2  3 3 2 1 1 3 0 2 3
             *   Alt1_x1 Alt1_x3  3 3 2 2 2 1 1 2 2
             *   Alt1_x1 Alt2_x1  0 4 4 2 0 3 2 3 0
             *   Alt1_x1 Alt2_x2  3 3 2 2 1 2 3 1 1
             *   Alt1_x1 Alt2_x3  3 3 2 2 1 2 1 1 3

             *   Alt1_x2 Alt1_x3  2 2 0 3 1 2 1 4 3
             *   Alt1_x2 Alt2_x1  1 2 1 1 3 2 2 2 4
             *   Alt1_x2 Alt2_x2  0 2 2 3 0 3 5 3 0
             *   Alt1_x2 Alt2_x3  1 1 2 3 2 1 2 2 4

             *   Alt1_x3 Alt2_x1  1 3 2 2 2 3 1 2 2
             *   Alt1_x3 Alt2_x2  3 2 1 3 1 3 2 2 1
             *   Alt1_x3 Alt2_x3  0 3 3 3 0 4 3 2 0
```

```
*    Alt2_x1 Alt2_x2    1 1 2 3 3 1 4 1 2
*    Alt2_x1 Alt2_x3    1 2 1 3 1 3 2 2 3
*    Alt2_x2 Alt2_x3    1 2 5 2 2 1 3 1 1
     N-Way              1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Note that in this example, the input is a choice design (as opposed to the linear version of a choice design) so the results are in choice design format. There is one row for each alternative of each choice set. Some of the results are as follows:

```
                    Block    Set    Alt    x1    x2    x3

                      1       1      1      3     2     3
                                     2      2     3     1

                      1       2      1      2     1     2
                                     2      1     3     1


                      .
                      .
                      .
                    Block    Set    Alt    x1    x2    x3

                      2       1      1      2     2     1
                                     2      1     3     2

                      2       2      1      2     3     1
                                     2      3     1     3


                      .
                      .
                      .
```

%MktBlock Macro Options

The following options can be used with the `%MktBlock` macro:

Option	Description
`help`	(positional) “help” or “?” displays syntax summary
`alt=`*variable*	alternative number variable
`block=`*variable*	block number variable
`data=`*SAS-data-set*	either the choice design or linear arrangement
`factors=`*variable-list*	factors in the design
`id=`*variable-list*	variables to copy to output data set
`initblock=`*variable*	initial blocking variable

Option	Description
`iter=`*n*	times to try to block the design
`list=`*n*	list larger canonical correlations
`maxiter=`*n*	times to try to block the design
`nalts=`*n*	number of alternatives in choice set
`nblocks=`*n*	number of blocks to create
`next=`*n*	where to look for the next exchange
`options=nosort`	do not sort the design into blocks
`out=`*SAS-data-set*	output data set with block numbers
`outr=`*SAS-data-set*	randomized output data set
`print=`*print-options*	output display options
`ridge=`*n*	ridging factor
`seed=`*n*	random number seed
`set=`*variable*	choice set number variable
`vars=`*variable-list*	factors in the design

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%mktblock(help)
%mktblock(?)
```

alt= *variable*

specifies the alternative number variable. If this variable is in the input data set, it is excluded from the factor list. The default is `alt=Alt`.

block= *variable*

specifies the block number variable. If this variable is in the input data set, it is excluded from the factor list. The default is `block=Block`.

data= *SAS-data-set*

specifies either the choice design or the linear arrangement. The choice design has one row for each alternative of each choice set and one column for each of the attributes. Typically, this design is produced by either the `%ChoicEff` or `%MktRoll` macro. For choice designs, you must also specify the `nalts=` option. By default, the macro uses the last data set created. The linear arrangement has one row for each choice set and one column for each attribute of each alternative. Typically, this design is produced by the `%MktEx` macro. This is the design that is input into the `%MktRoll` macro.

factors= *variable-list*
vars= *variable-list*

specifies the factors in the design. By default, all numeric variables are used, except variables with names matching those specified in the `block=`, `set=`, `alt=`, and `id=` options. (By default, the variables `Block`, `Set`, `Run`, and `Alt` are excluded from the factor list.) If you are using version 8.2 or an earlier SAS release with a branded choice design (assuming the brand factor is called `Brand`), specify `id=Brand`. Do not add the brand factor to the factor list unless you are using SAS 9.0 or a later SAS release.

id= *variable-list*
specifies the `data=` data set variables to copy to the output data set. If you are using version 8.2 or an earlier SAS release with a branded choice design (assuming the brand factor is called `Brand`), specify `id=Brand`. Do not add the brand factor to the factor list unless you are using SAS 9.0 or a later SAS release.

initblock= *variable*
specifies the name of the variable in the data set that is to be used as the initial blocking variable for the first iteration.

list= *r*
lists canonical correlations larger than `list=`r. The default is $r = 0.316 \approx \sqrt{r^2 = 0.1}$.

maxiter= *n*
iter= *n*
specifies the number of times to try to block the design starting with a different random blocking. By default, the macro tries five random starts, and iteratively refines each until D-efficiency quits improving, then in the end selects the blocking with the best D-efficiency.

nalts= *n*
specifies the number of alternatives in each choice set. If you are inputting a choice design, you must specify `nalts=`, otherwise the macro assumes you are inputting a linear arrangement.

nblocks= *n*
specifies the number of blocks to create. The option `nblocks=1` just reports information about the design. The `nblocks=` option must be specified.

next= *n*
specifies how far into the design to go to look for the next exchange. The specification `next=1` specifies that the macro should try exchanging the level for each run with the level for the next run and all other runs. The specification `next=2` considers exchanges with half of the other runs, which makes the algorithm run more quickly. The macro considers exchanging the level for run i with run $i + 1$ then uses the `next=` value to find the next potential exchanges. Other values, including nonintegers can be specified as well. For example `next=1.5` considers exchanging observation 1 with observations 2, 4, 5, 7, 8, 10, 11, and so on. With smaller values, the macro tends to find a slightly better blocking variable at a cost of much slower run time.

options= *options-list*
specifies binary options. By default, no binary options are specified. Specify the following value after `options=`.

`nosort`
do not sort the design into blocks. This is useful anytime you want the order of the observations in the output data set to match the order of the observations in the input data set. You will typically not want to specify `options=nosort` when you are using the `%MktBlock` macro to block a design. However, `options=nosort` is handy when you are using the `%MktBlock` macro to add just another factor to the design.

out= *SAS-data-set*

specifies the output data set with the block numbers. The default is `out=blocked`. Often, you will want to specify a two-level name to create a permanent SAS data set so the design is available later for analysis.

outr= *SAS-data-set*

specifies the randomized output data set if you would like the design randomly sorted within blocks. Often, you will want to specify a two-level name to create a permanent SAS data set so the design is available later for analysis.

print= *print-options*

specifies both the `%MktBlock` and the `%MktEval` macro display options, which control the display of the results. The default is `print=normal`. Specify one or more values from the following list.

`all`	all output is displayed
`corr`	canonical correlations
`block`	canonical correlations within blocks
`design`	blocked design
`freqs`	long frequencies list
`list`	list of big canonical correlations
`nonzero`	like `ordered` but sets `list=1e-6`
`noprint`	no output is displayed
`normal`	`corr list summ design note`
`note`	blocking note
`ordered`	like `list` but ordered by variable names
`short`	`corr summ note`
`summ`	frequency summaries

ridge= *n*

specifies the value to add to the diagonal of $\mathbf{X}'\mathbf{X}$ to make it nonsingular. Usually, you will not need to change this value. If you do, you probably will not notice any effect. Specify `ridge=0` to use a generalized inverse instead of ridging. The default is `ridge=0.01`.

seed= *n*

specifies the random number seed. By default, `seed=0`, and clock time is used to make the random number seed. By specifying a random number seed, results should be reproducible within a SAS release for a particular operating system and for a particular version of the macro. However, due to machine and macro differences, some results might not be exactly reproducible everywhere, although you would expect the efficiency differences to be slight.

set= *variable*
specifies the choice set number variable. When `nalts=` is specified, the default is `Set`, otherwise the default is `Run`. If this variable is in the input data set, it is excluded from the factor list.

%MktBlock Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktBSize Macro

The `%MktBSize` autocall macro suggests sizes for balanced incomplete block designs (BIBDs). The sizes that it reports are sizes that meet necessary but not sufficient conditions for the existence of a BIBD, so a BIBD might not exist for every size reported. In the following example, a list of designs with 12 treatments, between 4 and 8 treatments per block, and between 12 and 30 blocks are requested:

```
   %mktbsize(t=12, k=4 to 8, b=12 to 30)
```

The results are as follows:

```
                                                                                   n
           t              k           b              r            Lambda       Total
      Number of       Block      Number        Treatment       Pairwise        Sample
      Treatments      Size       of Blocks     Frequency       Frequencies     Size

          12             6          22             11             5              132
```

You can use this information to create a BIBD with the `%MktBIBD` macro as follows:

```
   %mktbibd(t=12, k=6, b=22, seed=104)
```

There is no guarantee that `%MktBIBD` will find a BIBD for any specification, but in this case it does, and it finds the following design:

```
                        Balanced Incomplete Block Design

                        x1    x2    x3    x4    x5    x6

                         7     9     3    10    12     2
                         5     2     4     6    10     7
                         5    10    12     9    11     8
                         7     9    11    12     1     4
                        12     3     4     5     9    10
                        10     6     9    11     2     1
                         3     8     7     1     9     6
                         2    10    12     6     8     1
                         9     7     6     4     8     5
                         1    12     5     7     6    11
                        12     1     2     3     5     7
                         8     4     1    10     7    12
                         6     2     9     8    12     4
                         4     7    11     3    10     6
                        11     5     1     2     4     9
                        11     3     2     8     7     9
                         6    11     8     5     3    12
```

```
                     3      6     10      9      1      5
                     2      4      6     12     11      3
                     1      8     10     11      4      3
                     4      1      3      2      5      8
                    10      5      8      7      2     11
```

The design has b=22 blocks (rows), k=6 treatments per block (columns), and t=12 treatments (the entries are the integers 1 to 12). Each of the t=12 treatments occurs r=11 times, and each treatment occurs in a block with every other treatment λ=5 times.

The following step creates a list of over 50 designs:

```
%mktbsize(t=5 to 20, k=3 to t - 1, b=t to 30)
```

Some of the results are as follows:

```
                                                                       n
        t             k           b              r             Lambda       Total
    Number of       Block       Number         Treatment      Pairwise      Sample
    Treatments      Size        of Blocks      Frequency      Frequencies   Size

        5             3           10              6              3            30
        5             4            5              4              3            20
        6             3           10              5              2            30
        6             4           15             10              6            60
        6             5            6              5              4            30
```

Note that by default, `maxreps=1` (the maximum number of replications is 1), so for example, the design t=5, k=3, b=20 is not listed since it consists of two replications of t=5, k=3, b=10, which is listed. Also note that b, the number of blocks, was specified so that it is never less than the number of treatments. Furthermore, k, the block size (number of treatments per block), is set to always be less than the number of treatments. Even more complicated expressions are permitted. For example, to limit the number of treatments per block to no more than half of the number of treatments, you could specify the following:

```
%mktbsize(t=2 to 10, k=2 to 0.5 * t, b=t to 10)
```

The results are as follows:

```
                                                                   n
       t             k            b              r           Lambda       Total
   Number of      Block       Number         Treatment      Pairwise      Sample
   Treatments     Size        of Blocks      Frequency      Frequencies   Size

        4            2             6              3              1           12
        5            2            10              4              1           20
        6            3            10              5              2           30
        7            3             7              3              1           21
```

To limit the number of blocks as a function of the number of treatments, you could specify the following:

```
%mktbsize(t=2 to 10, k=2 to t - 1, b=t to 2 * t)
```

However, if you want to limit the number of treatments as a function of the number of blocks, you need to use the `order=` option to ensure that the number of blocks loop comes first, for example, as follows:

```
%mktbsize(b=2 to 10, t=2 to 0.5 * b, k=2 to t - 1, order=btk)
```

The macro reports sizes in which $r = b \times k/t$ and $l = r \times (k - 1)/(t - 1)$ are integers, $2 \leq k < t$, and $b \geq t$. When $r = b \times k/t$ and $l = r \times (k - 1)/(t - 1)$ are integers, and $k = t$ and $b \geq t$, then a complete block design might be possible. This is a necessary but not sufficient condition for the existence of a complete block design. When $r = b \times k/t$ and $l = r \times (k - 1)/(t - 1)$ are integers, and $k < t$ and $b \geq t$, then a balanced incomplete block design might be possible. This is a necessary but not sufficient condition for the existence of a BIBD. When you specify `options=ubd` and $r = b \times k/t$ is an integer, then unbalanced block design sizes are reported as well. For example, if you want a design with `t=20` treatments and a block size of 6, you can run the following to find out how many blocks you need:

```
%mktbsize(t=20, k=6, options=ubd)
```

The results are as follows:

```
                                                                   n
       t             k            b              r          Lambda        Total
   Number of      Block       Number         Treatment      Pairwise      Sample
   Treatments     Size        of Blocks      Frequency      Frequencies   Size

       20            6            10              3            0.79          60
```

Then the `%MktBIBD` macro can be used to find a design where each treatment occurs 3 times, but the treatments do not appear together an equal number of times, for example, as follows:

```
%mktbibd(t=20, k=6, b=10, seed=104)
```

Some of the results are as follows:

```
                         Treatment by Treatment Frequencies

       1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20

  1    3   1   0   0   2   1   1   1   0   1   1   1   0   2   0   1   0   1   1   1
  2        3   0   1   1   1   1   1   1   1   0   1   1   0   1   0   1   0   2   1
  3            3   1   1   1   1   0   1   0   1   0   1   1   1   2   2   1   1   0
  4                3   0   1   1   1   0   1   1   1   1   0   1   1   1   1   2   0
  5                    3   1   1   1   1   1   0   1   1   1   0   2   0   0   1   0
  6                        3   0   0   1   1   1   0   1   1   1   1   0   1   1   1
  7                            3   1   0   0   2   2   1   1   1   0   1   0   1   0
  8                                3   1   1   1   1   2   1   0   1   1   0   0   1
  9                                    3   1   0   1   2   0   1   1   1   1   0   2
 10                                        3   0   1   0   1   2   1   0   1   1   1
 11                                            3   1   1   2   1   0   1   1   0   1
 12                                                3   1   0   1   0   0   1   1   1
 13                                                    3   0   0   1   1   0   0   1
 14                                                        3   1   1   1   1   0   1
 15                                                            3   0   1   1   1   1
 16                                                                3   1   1   1   0
 17                                                                    3   1   1   1
 18                                                                        3   1   2
 19                                                                            3   0
 20                                                                                3
```

%MktBSize Macro Options

The following options can be used with the `%MktBIBD` macro:

Option	Description
`help`	(positional) "help" or "?" displays syntax summary
`b=`*do-list*	number of blocks (alias for `nsets=`)
`k=`*do-list*	block size (alias for `setsize=`)
`maxreps=`*n*	maximum number of replications
`nattrs=`*do-list*	number of attributes (alias for `t=`)
`nsets=`*do-list*	number of sets (alias for `b=`)
`order=`*order-list*	order of the loops
`options=nocheck`	suppress checking b, t, and k
`options=ubd`	lifts the balance restriction on the design
`out=`*SAS-data-set*	output data set design list
`setsize=`*do-list*	set size (alias for `k=`)
`t=`*do-list*	number of treatments (alias for `nattrs=`)

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%mktbsize(help)
%mktbsize(?)
```

The `k=` or `setsize=`, and the `t=` or `nattrs=` options must be specified.

b= *do-list*
nsets= *do-list*

specifies the number of blocks. In a partial-profile design, this is the number of profiles. In a MaxDiff design, this is the number of sets. Specify either an integer or a list of integers in the SAS *do-list* syntax. The default is `b=2 to 500`. The `nsets=` and `b=` options are aliases.

k= *do-list*
setsize= *do-list*

specifies the block size, or the number of treatments in each block. In a partial-profile or MaxDiff design, this is the number of attributes or messages shown at one time in each set. Specify either an integer or a list of integers in the SAS *do-list* syntax. The `setsize=` and `k=` options are aliases. This option (in one of its two forms) must be specified.

maxreps= *n*

specifies the maximum number of replications. The default is `maxreps=1`. By default, this option prevents the `%MktBSize` macro from reporting designs of size $2b, 3b$, and so on after it has found a size with b blocks.

options= *options-list*

specifies binary options. By default, none of these options are specified. Specify one or more of the following values after `options=`.

`nocheck`
by default, certain checks are performed on b, t, and k. Specify `options=nocheck` to turn them off. This lets you make some creative expressions that otherwise would not be permitted.

`ubd`
lifts the balance restriction on the design. Results are reported when $r = b \times k/t$ is in integer but $l = r \times (k-1)/(t-1)$ might or might not be an integer. Use this option when you want to see sizes where every treatment can occur equally often, but the pairwise frequencies can be unequal. The listing can contain both sizes where a BIBD might be possible (λ, the expected pairwise frequency, is an integer) and sizes where a BIBD is not possible (λ is not an integer). You might use this option, for example, when the block design is being used to make a partial-profile design.

order= `tkb` | `tbk` | `btk` | `bkt` | `kbt` | `ktb`
specifies the order of the loops, the default is `tkb`, *t* then *k* then *b*. If you specify expressions in `t=`, `b=`, or `k=`, you might need some other ordering. For example, if you specify something like `t = 2 to 0.5 * b`, then you must specify `order=bkt` or any other ordering that defines *b* before *t*. Alternatively, you can specify this option just to change the default ordering of the results.

out= *SAS-data-set*
specifies the output data set with the list of potential design sizes. The default is `out=bibd`.

t= *do-list*
nattrs= *do-list*
specifies the number of treatments. In a partial-profile or MaxDiff design, this is the total number of attributes or messages. Specify either an integer or a list of integers in the SAS *do-list* syntax. The `nattrs=` and `t=` options are aliases. This option (in one of its two forms) must be specified. When the `nattrs=` option is specified, the output will use the word "Attribute" rather than "Treatment" and "Set" rather than "Block".

%MktBSize Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktDes Macro

The %MktDes autocall macro creates efficient experimental designs. Usually, we will not need to call the %MktDes macro directly. Instead, we will usually use the %MktEx autocall macro, which calls the %MktDes macro as one of its many tools. At the heart of the %MktDes macro are PROC PLAN, PROC FACTEX, and PROC OPTEX. PROC PLAN creates full-factorial designs. PROC FACTEX creates fractional-factorial designs. Both procedures can be used to create a candidate set for PROC OPTEX to search. We use a macro instead of calling these procedures directly because the macro has a simpler syntax. You specify the names of the factors and the number of levels for each factor. You also specify the number of runs you want in your final design. For example, you can create a design in 18 runs with 2 two-level factors (x1 and x2) and 3 three-level factors (x3, x4, and x5) as follows:

```
%mktdes(factors=x1-x2=2 x3-x5=3, n=18)
```

You can optionally specify interactions that you want to estimate. The macro creates a candidate design in which every effect you want to estimate is estimable, but the candidate design is bigger than you want. By default, the candidate set is stored in a SAS data set called CAND1. The macro then uses PROC OPTEX to search the candidate design for an efficient final design. By default, the final experimental design is stored in a SAS data set called DESIGN.

When the full-factorial design is small (by default less than 2189 runs, although sizes up to 5000 or 6000 runs are reasonably small), the experimental design problem is straightforward. First, the macro uses PROC PLAN to create a full-factorial candidate set. Next, PROC OPTEX searches the full-factorial candidate set. For very small problems (a few hundred candidates) PROC OPTEX will often find the optimal design, and for larger problems, it might not find *the* optimal design, but given sufficient iteration (for example, specify maxiter=100 or more) it will find very good designs. Run time will typically be a few seconds or a few minutes, but it could be longer. The following shows a typical example of using the %MktDes macro to find an optimal nonorthogonal design when the full-factorial design is small (108 runs):

```
*---2 two-level factors and 3 three-level factors in 18 runs---;
%mktdes(factors=x1-x2=2 x3-x5=3, n=18, maxiter=500)
```

When the full-factorial design is larger, the macro uses PROC FACTEX to create a fractional-factorial candidate set. In those cases, the methods found in the %MktEx macro usually make better designs than those found with the %MktDes macro.

PROC FACTEX

The primary reason that the %MktDes macro exists is to provide a front end to PROC FACTEX, although it additionally provides a front end to PROC OPTEX and PROC PLAN. PROC FACTEX is powerful and general, and it does much more than we use it for here. However, that power leads to a somewhat long and detailed syntax. The %MktDes macro can help by writing that syntax for you. It should be pointed out that the syntax that the %MktDes macro writes for PROC FACTEX is often more verbose than it needs to be. It is simply easier for it to always write out a verbose style of syntax than try to be clever and take short cuts. Since PROC FACTEX is such an integral component of the %MktDes macro and hence the %MktEx macro, we will spend some time here reviewing what PROC FACTEX does and also the kinds of problems it was not designed to handle.

PROC FACTEX provides a powerful facility for constructing certain fractional-factorial designs. Specifically, PROC FACTEX is designed to make fractional-factorial designs based on prime numbers and powers of prime numbers. For example, since 2 is prime, you can use PROC FACTEX to make a design 2^{2^p-1} in 2^p runs. Examples include: 2^3 in $2^2 = 4$ runs, 2^7 in $2^3 = 8$ runs, 2^{15} in $2^4 = 16$ runs, 2^{31} in $2^5 = 32$ runs, 2^{63} in $2^6 = 64$ runs, and so on. Since 3 is prime, you can use PROC FACTEX to make a design $3^{(3^p-1)/2}$ in 3^p runs. Examples include: 3^4 in $3^2 = 9$ runs, 3^{13} in $3^3 = 27$ runs, 3^{40} in $3^4 = 81$ runs, and so on. Since 4 is a power of a prime, you can use PROC FACTEX to make a design $4^{(4^p-1)/3}$ in 4^p runs. Examples include: 4^5 in $4^2 = 16$ runs, 4^{21} in $4^3 = 64$ runs, and so on. Since 5 is prime, you can use PROC FACTEX to make a design $5^{(5^p-1)/4}$ in 5^p runs. Examples include: 5^6 in $5^2 = 25$ runs, 5^{31} in $5^3 = 125$ runs, and so on.

When the number of runs is a power of 2, you can use PROC FACTEX to make designs that contain mixes of 2, 4, 8, and other power-of-two-level factors. Examples include: $2^4 4^1$ in $2^3 = 8$ runs, $2^{3(5-q)}4^q$ in $2^4 = 16$ runs for $q < 5$, $2^{13}4^{12}8^2$ in $2^6 = 64$ runs, and so on. When the number of runs is a power of 3, you can use PROC FACTEX to make designs that contain mixes of 3, 9, 27, and other power-of-three-level factors, and so on.

The `%MktEx` macro uses the `%MktDes` macro and PROC FACTEX for all of the cases described so far. PROC FACTEX is used both to make larger orthogonal arrays and to make all fractional-factorial candidate sets. There are many times, however, when PROC FACTEX cannot be used to make orthogonal arrays. You *cannot* use PROC FACTEX to make a design $6^{(6^p-1)/5}$ in 6^p runs since 6 is not a prime number. Even for designs that exist, like 6^3 in $6^2 = 36$ runs, you cannot use PROC FACTEX to make this design, because again, 6 is not a prime number. Similarly, you cannot use PROC FACTEX to make 2^{11} in 12 runs (12 is not a power of 2), 2^{19} in 20 runs (20 is not a power of 2), 3^7 in 18 runs (18 is not a power of 3), and so on. You have to use the `%MktEx` macro to get designs like these. You can use PROC FACTEX to create the design 6^3 in 64 runs by creating 3 eight-level factors in 64 runs and coding down. However, such a design, while orthogonal, would be nonoptimal both compared to 6^3 in 36 runs (which is 100% D-efficient) and compared to an optimal but nonorthogonal array 6^3 in 64 runs.

The `%MktDes` and `%MktEx` macros rely on PROC FACTEX to make many orthogonal arrays that `%MktEx` cannot otherwise make, and the `%MktEx` macro can make many designs that PROC FACTEX cannot make. However, for $2^p < 128$, and for many of the other orthogonal arrays not based on powers of 2, both the `%MktEx` macro and PROC FACTEX can make the same designs, but by using totally different approaches. PROC FACTEX does a computerized search, whereas the `%MktEx` macro develops a difference scheme. The `%MktEx` macro will usually use its own methods to make these orthogonal arrays when an orthogonal array was directly requested, but use PROC FACTEX to make the same orthogonal array when it is needed for a candidate set.

We discussed making mixed orthogonal arrays when the numbers of levels are all powers of the same prime (with mixes of 2's and 4's being the most common example). You can also use PROC FACTEX to create other mixes in several ways. You can create a mix of two-level and three-level factors by first creating a mix of two-level and four-level (or eight-level) factors and then coding down the four- or eight-level factors into three-level factors. This preserves orthogonality but leads to imbalance. You can instead create a design with just the two-level factors (in 2^p runs) and a separate design with just the three-level factors (in 3^q runs) and cross them (pair each of the 2^p runs with each of the 3^q runs) creating a design in $2^p \times 3^q$ runs. This preserves orthogonality and balance, but the designs tend to get big quite quickly, and other approaches to mixed orthogonal array creation provide more factors. The smallest example of this approach is $2^3 3^4$ in $2^2 \times 3^2 = 36$ runs. In contrast, the `%MktEx` macro can directly construct the mixed orthogonal array $2^{11}3^{12}$ in 36 runs. The `%MktEx` macro uses the `%MktDes` macro to construct candidate sets using this first method, but does not employ the second method.

In summary, PROC FACTEX provides powerful software for constructing fractional-factorial designs. The `%MktDes` macro provides a front end to this procedure that makes it easier to call. The `%MktEx` macro uses the `%MktDes` macro for the things it is best at, but not for other things.

%MktDes Macro Options

The following options can be used with the `%MktDes` macro:

Option	Description
`help`	(positional) "help" or "?" displays syntax summary
`big=`*n*	size of big candidate set
`cand=`*SAS-data-set*	candidate design
`classopts=`*options*	`class` statement options
`coding=`*name*	`coding=` option
`examine=`< I > < V >	matrices that you want to examine
`facopts=`*options*	PROC FACTEX statement options
`factors=`*factor-list*	factors and levels for each factor
`generate=`*options*	`generate` statement options
`interact=`*interaction-list*	interaction terms
`iter=`*n*	number of designs
`keep=`*n*	number of designs to keep
`maxiter=`*n*	number of designs
`method=`*name*	search method
`n=`*n* | SATURATED	number of runs
`nlev=`*n*	number of levels for pseudo-factors
`options=allcode`	shows all code
`options=check`	checks the `cand=` design
`options=nocode`	suppress the procedure code display
`otherfac=`*variable-list*	other factors
`otherint=`*terms*	multi-step interaction terms
`out=`*SAS-data-set*	output experimental design
`procopts=`*options*	PROC OPTEX statement options
`run=`*procedure-list*	list of procedures that can be run
`seed=`*n*	random number seed
`size=`*n* | MIN	candidate-set size
`step=`*n*	step number
`where=`*where-clause*	`where` clause

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%mktdes(help)
%mktdes(?)
```

big= *n*

specifies the size at which the candidate set is considered to be big. By default, `big=2188`. If the size of the full-factorial design is less than or equal to this size, and if PROC PLAN is in the `run=` list, the macro uses PROC PLAN instead of PROC FACTEX to create the candidate set. The default of 2188 is $\max(2^{11}, 3^7) + 1$. Specifying values as large as `big=6000` or even slightly larger is often reasonable. However, run time is slower as the size of the candidate set increases. The `%MktEx` macro coordinate-exchange algorithm will usually work better than a candidate-set search when the full-factorial design has more than several thousand runs.

cand= *SAS-data-set*

specifies the output data set with the candidate design (from PROC FACTEX or PROC PLAN). The default name is `Cand` followed by the step number, for example: `Cand1` for step 1, `Cand2` for step 2, and so on. You should only use this option when you are reading an external candidate set. When you specify `step=` values greater than 1, the macro assumes the default candidate set names, CAND1, CAND2, and so on, were used in previous steps. Specify just a data set name, no data set options.

classopts= *options*

specifies PROC OPTEX `class` statement options. The default, is `classopts=param=orthref`. You probably never want to change this option.

coding= *name*

specifies the PROC OPTEX `coding=` option. This option is usually not needed.

examine= < I > < V >

specifies the matrices that you want to examine. The option `examine=I` displays the information matrix, $\mathbf{X}'\mathbf{X}$; `examine=V` displays the variance matrix, $(\mathbf{X}'\mathbf{X})^{-1}$; and `examine=I V` displays both. By default, these matrices are not displayed.

facopts= *options*

specifies PROC FACTEX statement options.

factors= *factor-list*

specifies the factors and the number of levels for each factor. The `factors=` option must be specified. All other options are not required. The following shows a simple example of creating a design with 10 two-level factors:

```
   %mktdes(factors=x1-x10=2)
```

First, a factor list, which is a valid SAS variable list, is specified. The factor list must be followed by an equal sign and an integer, which gives the number of levels. Multiple lists can be specified. For example, to create 5 two-level factors, 5 three-level factors, and 5 five-level factors, specify the following:

```
   %mktdes(factors=x1-x5=2 x6-x10=3 x11-x15=5)
```

By default, this macro creates each factor in a fractional-factorial candidate set from a minimum number of pseudo-factors. Pseudo-factors are not output; they are used to create the factors of interest and then discarded. For example, with `nlev=2`, a three-level factor `x1` is created from 2 two-level pseudo-factors (_1 and _2) and their interaction by coding down:

```
(_1=1, _2=1) -> x1=1
(_1=1, _2=2) -> x1=2
(_1=2, _2=1) -> x1=3
(_1=2, _2=2) -> x1=1
```

This creates imbalance—the 1 level appears twice as often as 2 and 3. Somewhat better balance can be obtained by instead using three pseudo-factors. The number of pseudo-factors can be specified in parentheses after the number of levels, for example, as follows:

```
%mktdes(factors=x1-x5=2 x6-x10=3(3))
```

The levels 1 to 8 are coded down to 1 2 3 1 2 3 1 3, which is better balanced. The cost is candidate-set size might increase and efficiency might actually decrease. Some researchers are willing to sacrifice a little bit of efficiency in order to achieve better balance.

generate= *options*

specifies the PROC OPTEX `generate` statement options. By default, additional options are not added to the `generate` statement.

interact= *interaction-list*

specifies interactions that must be estimable. By default, no interactions are guaranteed to be estimable. Examples:

```
interact=x1*x2
interact=x1*x2 x3*x4*x5
interact=x1|x2|x3|x4|x5@2
interact=@2
```

The interaction syntax is like PROC GLM's and many of the other modeling procedures. It uses "`*`" for simple interactions (`x1*x2` is the interaction between `x1` and `x2`), "`|`" for main effects and interactions (`x1|x2|x3` is the same as `x1 x2 x1*x2 x3 x1*x3 x2*x3 x1*x2*x3`) and "`@`" to eliminate higher-order interactions (`x1|x2|x3@2` eliminates `x1*x2*x3` and is the same as `x1 x2 x1*x2 x3 x1*x3 x2*x3`). The specification "`@2`" creates main effects and two-way interactions. Only "`@`" values of `2` or `3` are permitted. If you specify "@2" by itself, a resolution V design is requested when PROC FACTEX is run.

iter= *n*
maxiter= *n*

specifies the PROC OPTEX `iter=` option which creates *n* designs. By default, `iter=10`.

keep= *n*

specifies the PROC OPTEX `keep=` option which keeps the *n* best designs. By default, `keep=5`.

nlev= *n*
specifies the number of levels from which factors are constructed through pseudo-factors and coding down. The value must be a prime or a power of a prime: 2, 3, 4, 5, 7, 8, 9, 11 .... This option is used with PROC FACTEX as follows:

```
factors factors / nlev=&nlev;
```

By default, the macro uses the minimum prime or power of a prime from the `factors=` list or 2 if no suitable value is found.

method= *name*
specifies the PROC OPTEX `method=` search method option. The default is `method=m_Fedorov` (modified Fedorov).

n= *n* | `saturated`
specifies the PROC OPTEX `n=` option, which is the number of runs in the final design. The default is the PROC OPTEX default and depends on the problem. Typically, you will not want to use the default. Instead, you should pick a value using the information produced by the `%MktRuns` macro as guidance (see page 1159). The `n=saturated` option creates a design with the minimum number of runs.

options= *options-list*
specifies binary options. By default, none of these options are specified. Specify one or more of the following values after `options=`.

`check`
checks the efficiency of a given design, specified in `cand=`.

`nocode`
suppresses the display of the PROC PLAN, PROC FACTEX, and PROC OPTEX code.

`allcode`
shows all code, even code that will not be run.

otherfac= *variable-list*
specifies other terms to mention in the `factors` statement of PROC FACTEX. These terms are not guaranteed to be estimable. By default, there are no other factors.

otherint= *terms*
specifies interaction terms that will only be specified with PROC OPTEX for multi-step macro invocations. By default, no interactions are guaranteed to be estimable. Normally, interactions that are specified via the `interact=` option affect both the PROC FACTEX and the PROC OPTEX `model` statements. In multi-step problems, part of an interaction might not be in a particular PROC FACTEX step. In that case, the interaction term must only appear in the PROC OPTEX step. For example, if `x1` is created in one step and `x4` is created in another, and if the `x1*x4` interaction must be estimable, specify `otherint=x1*x4` on the final step, the one that runs PROC OPTEX. The following steps create

the design:

```
%mktdes(step=1, factors=x1-x3=2, n=30, run=factex)

%mktdes(step=2, factors=x4-x6=3, n=30, run=factex)

%mktdes(step=3, factors=x7-x9=5, n=30, run=factex optex,
        otherint=x1*x4)
```

out= *SAS-data-set*
specifies the output experimental design (from PROC OPTEX). By default, `out=design`. Often, you will want to specify a two-level name to create a permanent SAS data set so the design is available later for analysis.

procopts= *options*
specifies PROC OPTEX statement options. By default, no options are added to the PROC OPTEX statement.

run= *procedure-list*
specifies the list of procedures that the macro can run. Normally, the macro runs either PROC FACTEX or PROC PLAN and then PROC OPTEX. By default, `run=plan factex optex`. You can skip steps by omitting procedure names from this list. When both PLAN and FACTEX are in the list, the macro chooses between them based on the size of the full-factorial design and the value of `big=`. When PLAN is not in the list, the macro generates code for PROC FACTEX.

seed= *n*
specifies the random number seed. By default, `seed=0`, and clock time is used to make the random number seed. By specifying a random number seed, results should be reproducible within a SAS release for a particular operating system and for a particular version of the macro. However, due to machine and macro differences, some results might not be exactly reproducible everywhere, although you would expect the efficiency differences to be slight.

size= *n* | `min`
specifies the candidate-set size. Start with the default `size=min` and see how big that design is. If you want, subsequently you can specify larger values that are `nlev=`*n* multiples of the minimum size. This option is used with PROC FACTEX as follows:

```
size design=&size;
```

When `nlev=`*n*, increase the `size=` value by a factor of *n* each time. For example, when `nlev=2`, increase the `size=` value by a factor of two each time. If `size=min` implies `size=128`, then 256, 512, 1024, and 2048 are reasonable sizes to try. Integer expressions like `size=128*4` are permitted.

step= *n*
specifies the step number. By default, there is only one step. However, sometimes, a better design can be found using a multi-step approach. Do not specify the `cand=` option in any step of a multi-step run. Consider the problem of making a design with 3 two-level factors, 3 three-level factors, and 3 five-level factors. The simplest approach is to do something like the following—create a design from two-level factors using pseudo-factors and coding down:

```
%mktdes(factors=x1-x3=2 x4-x6=3 x7-x9=5, n=30)
```

However, for small problems like this, the following three-step approach is usually better:

```
%mktdes(step=1, factors=x1-x3=2, n=30, run=factex)
%mktdes(step=2, factors=x4-x6=3, n=30, run=factex)
%mktdes(step=3, factors=x7-x9=5, n=30, run=factex optex)
```

Note, however, that the following `%MktEx` macro step will usually be better still:

```
%mktex(2 2 2 3 3 3 5 5 5, n=30)
```

The first `%MktDes` macro step uses PROC FACTEX to create a fractional-factorial design for the two-level factors. The second step uses PROC FACTEX to create a fractional-factorial design for the three-level factors and cross it with the two-level factors. The third step uses PROC FACTEX to create a fractional-factorial design for the five-level factors and cross it with the design for the two and three-level factors and then run PROC OPTEX.

Each step globally stores two macro variables (`&class1` and `&inter1` for the first step, `&class2` and `&inter2` for the second step, ...) that are used to construct the PROC OPTEX `class` and `model` statements. When `step` $>$ 1, variables from the previous steps are used in the `class` and `model` statements. In this example, the following PROC OPTEX code is created by step 3:

```
proc optex data=Cand3;
   class
      x1-x3
      x4-x6
      x7-x9
      / param=orthref;
   model
      x1-x3
      x4-x6
      x7-x9
      ;
   generate n=30 iter=10 keep=5 method=m_fedorov;
   output out=Design;
   run; quit;
```

This step uses the previously stored macro variables `&class1=x1-x3` and `&class2=x4-x6`.

where= *where-clause*
specifies a SAS `where` clause for the candidate design, which is used to restrict the candidates. By default, the candidate design is not restricted.

%MktDes Macro Notes

This macro specifies `options nonotes` throughout much of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktDups Macro

The `%MktDups` autocall macro detects duplicate choice sets and duplicate alternatives within generic choice sets. See the following pages for examples of using this macro in the design chapter: 147, 174, 198, and 206. Also see the following pages for examples of using this macro in the discrete choice chapter: 319, 368, 519, 564, 564, 567, 570, 576, 597, 607, 617, 628, 636, 645, 650, 654, 656, 659 and 662. Additional examples appear throughout this chapter. To illustrate, consider a simple experiment with these two choice sets. These choice sets are completely different and are not duplicates.

```
a b c       a b c
1 2 1       1 1 1
2 1 2       2 2 2
1 1 2       2 2 1
2 1 1       1 2 2
```

Now consider these two choice sets:

```
a b c       a b c
1 2 1       2 1 2
2 1 2       1 1 2
1 1 2       2 1 1
2 1 1       1 2 1
```

They are the same for a generic study because all of the same alternatives are there, they are just in a different order. However, for a branded study they are different. For a branded study, there would be a different brand for each alternative, so the choice sets would be the same only if all the same alternatives appeared in the same order. For both a branded and generic study, these choice sets are duplicates:

```
a b c       a b c
1 2 1       1 2 1
2 1 2       2 1 2
1 1 2       1 1 2
2 1 1       2 1 1
```

Now consider these choice sets for a generic study.

```
a b c       a b c
1 2 1       1 2 1
2 1 1       1 2 1
1 1 2       1 1 2
2 1 1       2 1 1
```

First, each of these choice sets has duplicate alternatives (2 1 1 in the first and 1 2 1 in the second). Second, these two choice sets are flagged as duplicates, even though they are not exactly the same. They are flagged as duplicates because every alternative in choice set one is also in choice set two, and every alternative in choice set two is also in choice set one. In generic studies, two choice sets are considered duplicates unless one has one or more alternatives that are not in the other choice set.

As an example, a design is created with the `%ChoicEff` macro choice-set-swapping algorithm for a branded study, then the `%MktDups` macro is run to check for and eliminate duplicate choice sets. The following steps create and evaluate the design:

```
%mktex(3 ** 9, n=27, seed=424)

data key;
   input (Brand x1-x3) ($);
   datalines;
Acme   x1 x2 x3
Ajax   x4 x5 x6
Widgit x7 x8 x9
;

%mktroll(design=randomized, key=key, alt=brand, out=cand)

%choiceff(data=cand,                /* candidate set of choice sets         */
          model=class(brand x1-x3 / sta),/* model with stdz orthog coding   */
          seed=420,                 /* random number seed                   */
          nsets=18,                 /* number of choice sets                */
          nalts=3,                  /* number of alternatives               */
          options=relative,         /* display relative D-efficiency        */
          beta=zero)                /* assumed beta vector, Ho: b=0         */

proc freq; tables set; run;

%mktdups(branded, data=best, factors=brand x1-x3, nalts=3, out=out)

proc freq; tables set; run;
```

The following PROC FREQ results show that candidate choice sets occur more than once in the design:

```
                            The FREQ Procedure

                                               Cumulative    Cumulative
                Set    Frequency     Percent    Frequency      Percent
                --------------------------------------------------------
                  1           3        5.56            3         5.56
                  2           6       11.11            9        16.67
                  3           3        5.56           12        22.22
                  5           6       11.11           18        33.33
                  8           3        5.56           21        38.89
                  9           3        5.56           24        44.44
                 12           3        5.56           27        50.00
                 13           3        5.56           30        55.56
                 14           6       11.11           36        66.67
                16              6       11.11               42          77.78
                19              3        5.56               45          83.33
                20              3        5.56               48          88.89
                24              3        5.56               51          94.44
                27              3        5.56               54         100.00
```

The `%MktDups` macro displays the following information in the log:

```
Design:          Branded
Factors:         brand x1-x3
                 Brand
                 x1 x2 x3
Duplicate Sets:  4
```

The output from the `%MktDups` macro contains the following table:

```
                                     Duplicate
                          Choice    Choice Sets
                             Set     To Delete

                               1            15

                               3            18

                               7            14

                              10            17
```

The first line of the first table tells us that this is a branded design as opposed to generic. The second line tells us the factors as specified in the `factors=` option. These are followed by the actual variable names for the factors. The last line reports the number of duplicates. The second table tells us that choice set 1 is the same as choice set 15. Similarly, 3 and 18 are the same, and so on. The `out=` data set will contain the design with the duplicate choice set eliminated.

Now consider an example with purely generic alternatives. The following steps create and evaluate the design:

```
%mktex(2 ** 5, n=2**5, seed=109)

%choiceff(data=randomized,          /* candidate set of alternatives       */
          model=class(x1-x5 / sta), /* model with stdz orthogonal coding   */
          seed=93,                  /* random number seed                  */
          nsets=42,                 /* number of choice sets               */
          flags=4,                  /* 4 alternatives, generic candidates  */
          options=relative,         /* display relative D-efficiency       */
          beta=zero)                /* assumed beta vector, Ho: b=0        */

%mktdups(generic, data=best, factors=x1-x5, nalts=4, out=out)
```

The macro produces the following tables:

```
Design:          Generic
Factors:         x1-x5
                 x1 x2 x3 x4 x5
Sets w Dup Alts: 1
Duplicate Sets:  1
```

```
                      Duplicate
          Choice    Choice Sets
             Set      To Delete

               2             25

              39                        Alternatives
```

For each choice set listed in the choice set column, either the other choice sets it duplicates are listed or the word `Alternatives` is displayed if the problem is with duplicate alternatives.

The following step displays just the choice sets with duplication problems:

```
proc print data=best;
   var x1-x5;
   id set; by set;
   where set in (2, 25, 39);
   run;
```

The results are as follows:

```
Set    x1    x2    x3    x4    x5

  2     1     2     1     1     1
        2     2     1     1     1
        1     1     2     2     2
        2     1     2     2     2

 25     1     1     2     2     2
        2     1     2     2     2
        2     2     1     1     1
        1     2     1     1     1

 39     1     1     2     1     1
        1     1     2     1     1
        2     2     1     2     2
        2     2     1     2     2
```

You can see that the macro detects duplicates even though the alternatives do not always appear in the same order in the different choice sets.

Now consider another example. The following steps create and evaluate a choice design:

```
%mktex(2 ** 6, n=2**6)

%mktroll(design=design, key=3 2, out=cand)

%mktdups(generic, data=cand, factors=x1-x2, nalts=3, out=out)

proc print; by set; id set; run;
```

Some of the results are as follows:

```
Design:          Generic
Factors:         x1-x2
                 x1 x2
Sets w Dup Alts: 40
Duplicate Sets:  50
```

```
                       Duplicate
             Choice   Choice Sets
                Set    To Delete

                  1                 Alternatives

                  2                 Alternatives
                               5
                               6
                              17
                              18
                              21
                  .
                  .
                  .
```

The output lists, for each set of duplicates, the choice set that is kept (in the first column) and all the matching choice sets that are deleted (in the second column).

The unique choice sets are as follows:

```
Set    _Alt_    x1    x2

  7      1       1     1
         2       1     2
         3       2     1

  8      1       1     1
         2       1     2
         3       2     2
```

```
                                    12      1       1     1
                                            2       2     1
                                            3       2     2

                                    28      1       1     2
                                            2       2     1
                                            3       2     2
```

The following example creates a conjoint design[*] and tests it for duplicates:

```
   %mktex(3 ** 3 2 ** 2, n=19, seed=513)

   %mktdups(linear, data=design, factors=x1-x5)
```

The results are as follows:

```
  Design:          Linear
  Factors:         x1-x5
                   x1 x2 x3 x4 x5
  Duplicate Runs:  2
```

```
                                              Duplicate
                                                   Runs
                                     Run      To Delete

                                      12             13

                                      16             17
```

%MktDups Macro Options

The following options can be used with the `%MktDups` macro:

Option	Description
`options`	(positional) binary options
	(positional) “help” or “?” displays syntax summary
`data=`*SAS-data-set*	input choice design
`factors=`*variable-list*	factors in the design
`nalts=`*n*	number of alternatives
`out=`*SAS-data-set*	output data set
`outlist=`*SAS-data-set*	output data set with duplicates
`vars=`*variable-list*	factors in the design

[*]Normally, we would use 18 runs and not a prime number like 19 that is not divisible by any of the numbers of levels, 2 and 3. We picked a silly number like 19 to ensure duplicates for this example.

Help Option

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%mktdups(help)
%mktdups(?)
```

Positional Parameter

The options list is a positional parameter. This means it must come first, and unlike all other parameters, it is not specified after a name and an equal sign.

options

specifies positional options. You can specify `noprint` and one of the following: `generic`, `branded`, or `linear`.

`branded`
specifies that since one of the factors is brand, the macro only needs to compare corresponding alternatives in each choice set.

`generic`
specifies a generic design and is the default. This means that there are no brands, so options are interchangeable, so the macro needs to compare each alternative with every other alternative in every choice set.

`linear`
specifies a linear not a choice design. Specify `linear` for a full-profile conjoint design, for an ANOVA design, or for the linear version of a branded choice design.

`noprint`
suppresses the output display. This option is used when you are only interested in the output data set or macro variable.

The following step shows an example of the `noprint` option:

```
%mktdups(branded noprint, data=design, nalts=3)
```

Required Options

This next option is mandatory with choice designs.

nalts= *n*

specifies the number of alternatives. This option must be specified with generic or branded choice designs. It is ignored with linear model designs. For generic or branded designs, the `data=` data set must contain `nalts=` observations for the first choice set, `nalts=` observations for the second choice set, and so on.

Other Options

The other options are as follows:

data= *SAS-data-set*
specifies the input choice design. By default, the macro uses the last data set created.

out= *SAS-data-set*
specifies an output data set that contains the design with duplicate choice sets excluded. By default, no data set is created, and the macro just reports on duplicates. Often, you will want to specify a two-level name to create a permanent SAS data set so the design is available later for analysis.

outlist= *SAS-data-set*
specifies the output data set with the list of duplicates. By default, `outlist=outdups`.

vars= *variable-list*
factors= *variable-list*
specifies the factors in the design. By default, all numeric variables are used.

%MktDups Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktEval Macro

The %MktEval autocall macro evaluates an experimental design for a linear model. The %MktEval macro reports on balance and orthogonality. Typically, you will call it immediately after running the %MktEx macro. You do not call it after making a choice design by using the %ChoicEff macro. The descriptive statistics that the %MktEval macro produces are appropriate for linear models not choice models. However, you can reasonably call it with the linear arrangement that will later be transformed into a choice design, for example, with the %MktRoll macro. See page 130 for an example of using this macro in the design chapter. Also see the following pages for examples of using this macro in the discrete choice chapter: 306, 308, 349, 353, 413, 423, 480, 485, 489, 491, 493, 538, 588 and 591. Additional examples appear throughout this chapter.

The output from this macro contains two default tables. The first table shows the canonical correlations between pairs of coded factors. A canonical correlation is the maximum correlation between linear combinations of the coded factors. See page 101 for more information about canonical correlations. All zeros off the diagonal show that the design is orthogonal for main effects. Off-diagonal canonical correlations greater than 0.316 ($r^2 > 0.1$) are listed in a separate table.

For nonorthogonal designs and designs with interactions, the canonical-correlation matrix is not a substitute for looking at the variance matrix with the %MktEx macro. It just provides a quick and more-compact picture of the correlations between the factors. The variance matrix is sensitive to the actual model specified and the coding. The canonical-correlation matrix just tells you if there is some correlation between the main effects. When is a canonical correlation too big? You will have to decide that for yourself. In part, the answer depends on the factors and how the design will be used. A high correlation between the client's and the main competitor's price factor is a serious problem meaning you will need to use a different design. In contrast, a moderate correlation in a choice design between one brand's minor attribute and another brand's minor attribute might be perfectly fine.

The macro also displays one-way, two-way and n-way frequencies. Equal one-way frequencies occur when the design is balanced. Equal two-way frequencies occur when the design is orthogonal. Equal n-way frequencies, all equal to one, occur when there are no duplicate runs or choice sets.

The following steps create and evaluate a design:

```
%mktex(2 2 3 ** 6,                  /* 2 two-level and 6 three-level factors*/
       n=18,                        /* 18 runs                              */
       balance=0,                   /* require perfect balance in the end   */
       mintry=5*18,                 /* but imbalance OK for first 5 passes  */
       seed=289)                    /* random number seed                   */

%mkteval(data=randomized)
```

The results are as follows:

```
                 Canonical Correlations Between the Factors
             There is 1 Canonical Correlation Greater Than 0.316

             x1      x2      x3      x4      x5      x6      x7      x8

       x1    1       0.33    0       0       0       0       0       0
       x2    0.33    1       0       0       0       0       0       0
       x3    0       0       1       0       0       0       0       0
       x4    0       0       0       1       0       0       0       0
       x5    0       0       0       0       1       0       0       0
       x6    0       0       0       0       0       1       0       0
       x7    0       0       0       0       0       0       1       0
       x8    0       0       0       0       0       0       0       1

              Canonical Correlations > 0.316 Between the Factors
             There is 1 Canonical Correlation Greater Than 0.316


                                   r    r Square

                        x1    x2    0.33      0.11

                          Summary of Frequencies
             There is 1 Canonical Correlation Greater Than 0.316
                      * - Indicates Unequal Frequencies

                          Frequencies

                   x1       9 9
                   x2       9 9
                   x3       6 6 6
                   x4       6 6 6
                   x5       6 6 6
                   x6       6 6 6
                   x7       6 6 6
                   x8       6 6 6
              *    x1 x2    3 6 6 3
                   x1 x3    3 3 3 3 3 3
                   x1 x4    3 3 3 3 3 3
                   x1 x5    3 3 3 3 3 3
                   x1 x6    3 3 3 3 3 3
                   x1 x7    3 3 3 3 3 3
                   x1 x8    3 3 3 3 3 3
                   x2 x3    3 3 3 3 3 3
                   x2 x4    3 3 3 3 3 3
                   x2 x5    3 3 3 3 3 3
                   x2 x6    3 3 3 3 3 3
                   x2 x7    3 3 3 3 3 3
                   x2 x8    3 3 3 3 3 3
```

```
x3 x4     2 2 2 2 2 2 2 2 2
x3 x5     2 2 2 2 2 2 2 2 2
x3 x6     2 2 2 2 2 2 2 2 2
x3 x7     2 2 2 2 2 2 2 2 2
x3 x8     2 2 2 2 2 2 2 2 2
x4 x5     2 2 2 2 2 2 2 2 2
x4 x6     2 2 2 2 2 2 2 2 2
x4 x7     2 2 2 2 2 2 2 2 2
x4 x8     2 2 2 2 2 2 2 2 2
x5 x6     2 2 2 2 2 2 2 2 2
x5 x7     2 2 2 2 2 2 2 2 2
x5 x8     2 2 2 2 2 2 2 2 2
x6 x7     2 2 2 2 2 2 2 2 2
x6 x8     2 2 2 2 2 2 2 2 2
x7 x8     2 2 2 2 2 2 2 2 2
N-Way     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

All factors in this design are perfectly balanced, and almost all are orthogonal, but `x1` and `x2` are correlated with each other.

%MktEval Macro Options

The following options can be used with the `%MktEval` macro:

Option	Description
`help`	(positional) "help" or "?" displays syntax summary
`blocks=`*variable*	blocking variable
`data=`*SAS-data-set*	input data set with design
`factors=`*variable-list*	factors in the design
`format=`*format*	format for canonical correlations
`freqs=`*frequency-list*	frequencies to display
`list=`*n*	minimum canonical correlation to list
`outcb=`*SAS-data-set*	within-block canonical correlations
`outcorr=`*SAS-data-set*	canonical correlation matrix
`outfreq=`*SAS-data-set*	frequencies
`outfsum=`*SAS-data-set*	frequency summaries
`outlist=`*SAS-data-set*	list of largest canonical correlations
`print=`*print-options*	controls the display of the results
`vars=`*variable-list*	factors in the design

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%mkteval(help)
%mkteval(?)
```

blocks= *variable*
specifies a blocking variable. This option displays separate canonical correlations within each block. By default, there is one block.

data= *SAS-data-set*
specifies the input SAS data set with the experimental design. By default, the macro uses the last data set created.

factors= *variable-list*
vars= *variable-list*
specifies a list of the factors in the experimental design. The default is all of the numeric variables in the data set.

freqs= *frequency-list*
specifies the frequencies to display. By default, `freqs=1 2 n`, and 1-way, 2-way, and n-way frequencies are displayed. Do not specify the exact number of ways instead of `n`. For ways other than `n`, the macro checks for and displays zero cell frequencies. For n-ways, the macro does not output or display zero frequencies. Only the full-factorial design will have nonzero cells, so specifying something like `freqs=1 2 20` will make the macro take a *long* time, and it will try to create *huge* data sets and will probably run out of memory or disk space before it is done. However, `freqs=1 2 n` runs very reasonably.

format= *format*
specifies the format for displaying the canonical correlations. The default format is `4.2`.

list= *n*
specifies the minimum canonical correlation to list. The default is 0.316, the square root of $r^2 = 0.1$.

outcorr= *SAS-data-set*
specifies the output SAS data set for the canonical correlation matrix. The default data set name is CORR.

outcb= *SAS-data-set*
specifies the output SAS data set for the within-block canonical correlation matrices. The default data set name is CB.

outlist= *SAS-data-set*
specifies the output data set for the list of largest canonical correlations. The default data set name is LIST.

outfreq= *SAS-data-set*
specifies the output data set for the frequencies. The default data set name is FREQ.

outfsum= *SAS-data-set*
specifies the output data set for the frequency summaries. The default data set name is FSUM.

print= *print-options*
controls the display of the results. The default is `print=short`. Specify one or more values from the following list.

`all`	all output is displayed
`corr`	displays the canonical correlations matrix
`block`	displays the canonical correlations within block
`freqs`	displays the frequencies, specified by the `freqs=` option
`list`	displays the list of canonical correlations greater than the `list=` value
`nonzero`	like `ordered` but sets `list=1e-6`
`ordered`	like `list` but ordered by variable names
`short`	is the default and is equivalent to: `corr list summ block`
`summ`	displays the frequency summaries
`noprint`	no output is displayed

By default, the frequency list, which contains the factor names, levels, and frequencies is not displayed, but the more compact frequency summary list, which contains the factors and frequencies but not the levels is displayed.

%MktEval Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktEx Macro

The `%MktEx` autocall macro creates efficient factorial designs. The `%MktEx` macro is designed to be very simple to use and to run in seconds for trivial problems, minutes for small problems, and in less than an hour for larger and difficult problems. This macro is a full-featured factorial-experimental designer that can handle simple problems like main-effects designs and more complicated problems including designs with interactions and restrictions on which levels can appear together. The macro is designed to easily create the kinds of designs that marketing researchers need for conjoint and choice experiments and that other researchers need for other types of problems. For most factorial-design problems, you can simply run the macro once, specifying only the number of runs and the numbers of levels of all the factors. You no longer have to try different algorithms and different approaches to see which one works best. The macro does all of that for you. We state on page 244 “The best approach to design creation is to use the computer as a tool along with traditional design skills, not as a substitute for thinking about the problem.” With the `%MktEx` macro, we try to automate some of the thought processes of the expert designer.

See the following pages for examples of using this macro in the design chapter: 81, 85, 98, 109, 112, 129, 166, 190 and 200. Also see the examples of using this macro in the discrete choice chapter from pages 285 through 663. Additional examples appear throughout this chapter.

The following example uses the `%MktEx` macro to create a design with 5 two-level factors, 4 three-level factors, 3 five-level factors, 2 six-level factors, all in 60 runs (row, experimental conditions, conjoint profiles, or choice sets):

```
   %mktex(2 ** 5  3 ** 4  5 5 5  6 6, n=60)
```

The notation `m ** n` means m^n or n m-level factors. For example `2 ** 5` means $2 \times 2 \times 2 \times 2 \times 2$ or 5 two-level factors.

The `%MktEx` macro creates efficient factorial designs using several approaches. The macro will try to directly create an orthogonal design (strength-two orthogonal array), it will search a set of candidate runs (rows of the design), and it will use a coordinate-exchange algorithm using both random initial designs and also a partially orthogonal design initialization. The macro stops if at any time it finds a perfect, 100% efficient, orthogonal and balanced design. This first phase is the algorithm search phase. In it, the macro determines which approach is working best for this problem. At the end of this phase, the macro chooses the method that has produced the best design and performs another set of iterations using exclusively the chosen approach. Finally, the macro performs a third set of iterations where it takes the best design it found so far and tries to improve it.

In all phases, the macro attempts to optimize D-efficiency (sometimes known as D-optimality), which is a standard measure of the goodness of the experimental design. As D-efficiency increases, the standard errors of the parameter estimates in the linear model decrease. A perfect design is orthogonal and balanced and has 100% D-efficiency. A design is orthogonal when all of the parameter estimates are uncorrelated. A design is balanced when all of the levels within each of the factors occur equally often. A design is orthogonal and balanced when the variance matrix, which is proportional to $(\mathbf{X}'\mathbf{X})^{-1}$, is diagonal, where $\mathbf{X}$ is a suitable orthogonal coding (see page 73) of the design matrix. See pages 53 and 243, for more information about efficient experimental designs.

For most problems, you only need to specify the levels of all the factors and the number of runs. For more complicated problems, you might need to also specify the interactions that you want to estimate or restrictions on which levels may not appear together. Other than that, you should not need any other options for most problems. This macro is not like other design tools that you have to tell what to do. With this macro, you just tell it what you want, and it figures out a good way to do it. For some problems, the sophisticated user, with a lot of work, might be able to adjust the options to come up with a better design. However, this macro should always produce a very good design with minimal effort for even the most unsophisticated users.

Orthogonal Arrays

The `%MktEx` macro has the world's largest catalog of strength-two (main effects) orthogonal arrays. In an orthogonal array, all estimable effects are uncorrelated. The orthogonal arrays are constructed using methods and arrays from a variety of sources, including: Addelman (1962a,b); Bose (1947); Colbourn and de Launey (1996); Dawson (1985); De Cock and Stufken (2000); de Launey (1986, 1987a,b); Dey (1985); Ehrlich (1964); Elliott and Butson (1966); Hadamard (1893); Hedayat, Sloane, and Stufken (1999); Hedayat and Wallis (1978); Kharaghania and Tayfeh-Rezaiea (2004); Kuhfeld (2005); Kuhfeld and Suen (2005); Kuhfeld and Tobias (2005); Nguyen (2005); Nguyen (2006); Nguyen (2006); Paley (1933); Pang, Zhang, and Liu (2004a); Pang and Zhang (2004b); Rao (1947); Seberry and Yamada (1992); Sloane (2004); Spence (1975a); Spence (1975b); Spence (1977a); Spence (1977b); Suen (1989a,b, 2003a,b,c); Suen and Kuhfeld (2005); Taguchi (1987); Turyn (1972); Turyn (1974); Wang (1996a,b); Wang and Wu (1989, 1991); Williamson(1944); Xu (2002); Yamada (1986); Yamada (1989); Zhang (2004–2006); Zhang, Duan, Lu, and Zheng (2002); Zhang, Lu and Pang(1999); Zhang, Pang and Wang (2001); Zhang, Weiguo, Meixia and Zheng (2004); and the SAS FACTEX procedure.

For all n's up through 448 that are a multiple of 4, and many n's beyond that, the `%MktEx` macro can construct orthogonal designs with up to $n - 1$ two-level factors. The two-level designs are constructed from Hadamard matrices (Hadamard 1893; Paley 1933; Williamson 1944; Hedayat, Sloane, and Stufken 1999). The `%MktEx` macro can construct these designs when n is a multiple of 4 and one or more of the following hold:

- $n \leq 448$ or $n = 580, 596, 604, 612, 724, 732, 756,$ or 1060
- $n - 1$ is prime
- $n/2 - 1$ is a prime power and $\text{mod}(n/2, 4) = 2$
- n is a power of 2 (2, 4, 8, 16, ...) times the size of a smaller Hadamard matrix that is available.

When n is a multiple of 8, the macro can create orthogonal designs with a small number (say m) four-level factors in place of $3 \times m$ of the two-level factors (for example, $2^{70}\ 4^3$ in 80 runs).

You can see the Hadamard matrix sizes that `%MktEx` has in its catalog by running the following steps:

```
%mktorth(maxlev=2)

proc print; run;
```

Alternatively, you can see more details by instead running the following steps:

```
data x;
   length Method $ 30;
   do n = 4 to 1000 by 4;
      HadSize = n; method = ' ';
      do while(mod(hadsize, 8) eq 0); hadsize = hadsize / 2; end;
      link paley;
      if method eq ' ' and hadsize le 256 and not (hadsize in (188, 236))
         then method = 'Williamson';
      else if hadsize in (188, 236, 260, 268, 292, 356, 404, 436, 596)
         then method = 'Turyn, Hedayat, Wallis';
      else if hadsize = 324                   then method = 'Ehlich';
      else if hadsize in (372, 612, 732, 756)  then method = 'Turyn';
      else if hadsize in (340, 580, 724, 1060) then method = 'Paley 2';
      else if hadsize in (412, 604)           then method = 'Yamada';
      else if hadsize = 428 then method = 'Kharaghania and Tayfeh-Rezaiea';
      if method = ' ' then do;
         do while(hadsize lt n and method eq ' ');
            hadsize = hadsize * 2;
            link paley;
            end;
         end;
      if method ne ' ' then do; Change = n - lag(n); output; end;
      end;
   return;
   paley:;
      ispm1 = 1; ispm2 = mod(hadsize / 2, 4) eq 2;
      h = hadsize - 1;
      do i = 3 to sqrt(hadsize) by 2 while(ispm1);
         ispm1 = mod(h, i);
         end;
      h = hadsize / 2 - 1;
      do i = 3 to sqrt(hadsize / 2) by 2 while(ispm2);
         ispm2 = mod(h, i);
         end;
      if      ispm1 then method = 'Paley 1';
      else if ispm2 then method = 'Paley 2';
      return;
   run;

options ps=2100;
proc print label noobs;
   label hadsize = 'Reduced Hadamard Matrix Size';
   var n hadsize method change;
   run;
```

Note, however, that the fact that a number appears in this program's listing, does not guarantee that your computer will have enough memory and other resources to create it.

The following table provides a summary of the parent and design sizes up through 513 runs that are available with the `%MktEx` macro:

Number of Runs	Parents		All Designs	
4– 50	87	11.87%	181	0.15%
51–100	185	25.24%	611	0.52%
101–127 129–143 145–150	152	20.74%	287	0.24%
128	21	2.86%	740	0.63%
144	16	2.18%	1,241	1.06%
151–200	43	5.87%	1,073	0.91%
201–250	41	5.59%	1,086	0.92%
251–255 257–300	35	4.77%	3,451	2.94%
256	2	0.27%	6,101	5.19%
301–350	37	5.05%	2,295	1.95%
351–400	41	5.59%	4,734	4.03%
401–431 433–447 449–450	24	3.27%	425	0.36%
432	7	0.95%	10,839	9.22%
448	4	0.55%	8,598	7.31%
451–500	32	4.37%	1,789	1.52%
501–511 513	4	0.55%	113	0.10%
512	2	0.27%	73,992	62.96%
	733		117,556	

Designs with 128, 144, 256, 432, 448, and 512 runs are listed separately from the rest of their category since there are so many of them.

Not included in this list are 2296 additional designs that are explicitly in the catalog but have more than 513 runs. Most are constructed from the parent array 24^8 in 576 runs (which is useful for making Latin Square designs). The rest are constructed from Hadamard matrices.

The 733 parent designs are displayed following this paragraph. (Listing all of the 117,556 designs at 200 designs per page, would require 588 pages.) Many more orthogonal arrays that are not explicitly in this catalog can also be created. These include full-factorial designs with more than 144 runs, Hadamard designs with more than 1000 runs, and fractional-factorial designs in 256, 512, or more runs.

Parents	Designs	Runs	
1	1	**4**	2^{3}
1	1	**6**	$2^{1}3^{1}$
1	2	**8**	$2^{4}4^{1}$
1	1	**9**	3^{4}
1	1	**10**	$2^{1}5^{1}$
4	4	**12**	2^{11}
			$2^{4}3^{1}$
			$2^{2}6^{1}$
			$3^{1}4^{1}$
1	1	**14**	$2^{1}7^{1}$
1	1	**15**	$3^{1}5^{1}$
2	7	**16**	$2^{8}8^{1}$
			4^{5}
2	3	**18**	$2^{1}9^{1}$
			$3^{6}6^{1}$
4	4	**20**	2^{19}
			$2^{8}5^{1}$
			$2^{2}10^{1}$
			$4^{1}5^{1}$
1	1	**21**	$3^{1}7^{1}$
1	1	**22**	$2^{1}11^{1}$
5	8	**24**	$2^{20}4^{1}$
			$2^{13}3^{1}4^{1}$
			$2^{12}12^{1}$
			$2^{11}4^{1}6^{1}$
			$3^{1}8^{1}$
1	1	**25**	5^{6}
1	1	**26**	$2^{1}13^{1}$
1	2	**27**	$3^{9}9^{1}$
4	4	**28**	2^{27}
			$2^{12}7^{1}$
			$2^{2}14^{1}$
			$4^{1}7^{1}$
3	4	**30**	$2^{1}15^{1}$
			$3^{1}10^{1}$
			$5^{1}6^{1}$
2	20	**32**	$2^{16}16^{1}$
			$4^{8}8^{1}$
1	1	**33**	$3^{1}11^{1}$
1	1	**34**	$2^{1}17^{1}$
1	1	**35**	$5^{1}7^{1}$
20	26	**36**	2^{35}
			$2^{27}3^{1}$
			$2^{20}3^{2}$
			$2^{18}3^{1}6^{1}$
			$2^{16}9^{1}$
			$2^{13}3^{2}6^{1}$
			$2^{13}6^{2}$
			$2^{10}3^{8}6^{1}$
			$2^{10}3^{1}6^{2}$
			$2^{9}3^{4}6^{2}$
			$2^{8}6^{3}$
			$2^{4}3^{1}6^{3}$
			$2^{3}3^{9}6^{1}$
			$2^{3}3^{2}6^{3}$
			$2^{2}3^{5}6^{2}$
			$2^{2}18^{1}$
			$2^{1}3^{3}6^{3}$
			$3^{12}12^{1}$
			$3^{7}6^{3}$
			$4^{1}9^{1}$
1	1	**38**	$2^{1}19^{1}$
1	1	**39**	$3^{1}13^{1}$
5	8	**40**	$2^{36}4^{1}$
			$2^{25}4^{1}5^{1}$
			$2^{20}20^{1}$
			$2^{19}4^{1}10^{1}$
			$5^{1}8^{1}$
3	4	**42**	$2^{1}21^{1}$
			$3^{1}14^{1}$
			$6^{1}7^{1}$
4	4	**44**	2^{43}
			$2^{15}11^{1}$
			$2^{2}22^{1}$
			$4^{1}11^{1}$
2	3	**45**	$3^{9}15^{1}$
			$5^{1}9^{1}$
1	1	**46**	$2^{1}23^{1}$
6	58	**48**	$2^{40}8^{1}$
			$2^{33}3^{1}8^{1}$
			$2^{31}6^{1}8^{1}$
			$2^{24}24^{1}$
			$3^{1}16^{1}$
			$4^{12}12^{1}$
1	1	**49**	7^{8}
2	3	**50**	$2^{1}25^{1}$
			$5^{10}10^{1}$
1	1	**51**	$3^{1}17^{1}$
4	4	**52**	2^{51}
			$2^{16}13^{1}$
			$2^{2}26^{1}$
			$4^{1}13^{1}$
3	6	**54**	$2^{1}27^{1}$
			$3^{20}6^{1}9^{1}$
			$3^{18}18^{1}$
1	1	**55**	$5^{1}11^{1}$
5	8	**56**	$2^{52}4^{1}$
			$2^{37}4^{1}7^{1}$
			$2^{28}28^{1}$
			$2^{27}4^{1}14^{1}$
			$7^{1}8^{1}$
1	1	**57**	$3^{1}19^{1}$
1	1	**58**	$2^{1}29^{1}$
11	15	**60**	2^{59}
			$2^{30}3^{1}$
			$2^{24}6^{1}$
			$2^{23}5^{1}$
			$2^{21}10^{1}$
			$2^{17}15^{1}$
			$2^{15}6^{1}10^{1}$
			$2^{2}30^{1}$
			$3^{1}20^{1}$
			$4^{1}15^{1}$
			$5^{1}12^{1}$
1	1	**62**	$2^{1}31^{1}$
2	3	**63**	$3^{12}21^{1}$
			$7^{1}9^{1}$
7	123	**64**	$2^{32}32^{1}$
			$2^{5}4^{17}8^{1}$
			$2^{5}4^{10}8^{4}$
			$4^{16}16^{1}$
			$4^{14}8^{3}$
			$4^{7}8^{6}$
			8^{9}
1	1	**65**	$5^{1}13^{1}$
3	4	**66**	$2^{1}33^{1}$
			$3^{1}22^{1}$
			$6^{1}11^{1}$
4	4	**68**	2^{67}
			$2^{18}17^{1}$
			$2^{2}34^{1}$
			$4^{1}17^{1}$
1	1	**69**	$3^{1}23^{1}$
3	4	**70**	$2^{1}35^{1}$
			$5^{1}14^{1}$
			$7^{1}10^{1}$

Parents	Designs	Runs	
49	103	**72**	$2^{68}4^{1}$
			$2^{60}3^{1}4^{1}$
			$2^{53}3^{2}4^{1}$
			$2^{51}3^{1}4^{1}6^{1}$
			$2^{49}4^{1}9^{1}$
			$2^{46}3^{2}4^{1}6^{1}$
			$2^{46}4^{1}6^{2}$
			$2^{44}3^{12}4^{1}$
			$2^{43}3^{8}4^{1}6^{1}$
			$2^{43}3^{1}4^{1}6^{2}$
			$2^{42}3^{4}4^{1}6^{2}$
			$2^{41}4^{1}6^{3}$
			$2^{37}3^{13}4^{1}$
			$2^{37}3^{1}4^{1}6^{3}$
			$2^{36}3^{9}4^{1}6^{1}$
			$2^{36}3^{2}4^{1}6^{3}$
			$2^{36}36^{1}$
			$2^{35}3^{12}4^{1}6^{1}$
			$2^{35}3^{5}4^{1}6^{2}$
			$2^{35}4^{1}18^{1}$
			$2^{34}3^{8}4^{1}6^{2}$
			$2^{34}3^{3}4^{1}6^{3}$
			$2^{31}6^{4}$
			$2^{30}3^{1}6^{4}$
			$2^{28}3^{2}6^{4}$
			$2^{27}3^{11}6^{1}12^{1}$
			$2^{27}3^{6}6^{4}$
			$2^{19}3^{20}4^{1}6^{1}$
			$2^{18}3^{16}4^{1}6^{2}$
			$2^{17}3^{12}4^{1}6^{3}$
			$2^{15}3^{7}4^{1}6^{5}$
			$2^{14}3^{3}4^{1}6^{6}$
			$2^{12}3^{21}4^{1}6^{1}$
			$2^{11}3^{20}6^{1}12^{1}$
			$2^{11}3^{17}4^{1}6^{2}$
			$2^{10}3^{20}4^{1}6^{2}$
			$2^{10}3^{16}6^{2}12^{1}$
			$2^{10}3^{13}4^{1}6^{3}$
			$2^{9}3^{16}4^{1}6^{3}$
			$2^{9}3^{12}6^{3}12^{1}$
			$2^{8}3^{12}4^{1}6^{4}$
			$2^{8}3^{8}4^{1}6^{5}$
			$2^{7}3^{7}6^{5}12^{1}$
			$2^{7}3^{4}4^{1}6^{6}$
			$2^{6}3^{7}4^{1}6^{6}$
			$2^{6}3^{3}6^{6}12^{1}$
			$2^{5}3^{3}4^{1}6^{7}$
			$3^{24}24^{1}$
			$8^{1}9^{1}$
1	1	**74**	$2^{1}37^{1}$
2	3	**75**	$3^{1}25^{1}$
			$5^{8}15^{1}$
4	4	**76**	2^{75}
			$2^{19}19^{1}$
			$2^{2}38^{1}$
			$4^{1}19^{1}$
1	1	**77**	$7^{1}11^{1}$
3	4	**78**	$2^{1}39^{1}$
			$3^{1}26^{1}$
			$6^{1}13^{1}$
7	49	**80**	$2^{72}8^{1}$
			$2^{61}5^{1}8^{1}$
			$2^{55}8^{1}10^{1}$
			$2^{51}4^{3}20^{1}$
			$2^{40}40^{1}$
			$4^{10}20^{1}$
			$5^{1}16^{1}$
2	12	**81**	$3^{27}27^{1}$
			9^{10}
1	1	**82**	$2^{1}41^{1}$
12	14	**84**	2^{83}
			$2^{33}3^{1}$
			$2^{28}7^{1}$
			$2^{27}6^{1}$
			$2^{22}6^{1}7^{1}$
			$2^{20}3^{1}14^{1}$
			$2^{20}21^{1}$
			$2^{14}6^{1}14^{1}$
			$2^{2}42^{1}$
			$3^{1}28^{1}$
			$4^{1}21^{1}$
			$7^{1}12^{1}$
1	1	**85**	$5^{1}17^{1}$
1	1	**86**	$2^{1}43^{1}$
1	1	**87**	$3^{1}29^{1}$
5	8	**88**	$2^{84}4^{1}$
			$2^{56}4^{1}11^{1}$
			$2^{44}44^{1}$
			$2^{43}4^{1}22^{1}$
			$8^{1}11^{1}$
5	10	**90**	$2^{1}45^{1}$
			$3^{30}30^{1}$
			$3^{26}6^{1}15^{1}$
			$5^{1}18^{1}$
			$9^{1}10^{1}$
1	1	**91**	$7^{1}13^{1}$
4	4	**92**	2^{91}
			$2^{21}23^{1}$
			$2^{2}46^{1}$
			$4^{1}23^{1}$
1	1	**93**	$3^{1}31^{1}$
1	1	**94**	$2^{1}47^{1}$
1	1	**95**	$5^{1}19^{1}$
14	183	**96**	$2^{80}16^{1}$
			$2^{73}3^{1}16^{1}$
			$2^{71}6^{1}16^{1}$
			$2^{48}48^{1}$
			$2^{44}4^{11}8^{1}12^{1}$
			$2^{43}4^{15}8^{1}$
			$2^{43}4^{12}6^{1}8^{1}$
			$2^{39}3^{1}4^{14}8^{1}$
			$2^{26}4^{23}$
			$2^{19}3^{1}4^{23}$
			$2^{18}4^{22}12^{1}$
			$2^{17}4^{23}6^{1}$
			$2^{12}4^{20}24^{1}$
			$3^{1}32^{1}$
2	3	**98**	$2^{1}49^{1}$
			$7^{14}14^{1}$
2	3	**99**	$3^{13}33^{1}$
			$9^{1}11^{1}$
15	23	**100**	2^{99}
			$2^{51}5^{3}$
			$2^{40}5^{4}$
			$2^{34}5^{3}10^{1}$
			$2^{29}5^{5}$
			$2^{22}25^{1}$
			$2^{18}5^{9}10^{1}$
			$2^{16}5^{3}10^{3}$
			$2^{7}5^{10}10^{1}$
			$2^{5}5^{4}10^{3}$
			$2^{4}10^{4}$
			$2^{2}50^{1}$
			$4^{1}25^{1}$
			$5^{20}20^{1}$
			$5^{8}10^{3}$
3	4	**102**	$2^{1}51^{1}$
			$3^{1}34^{1}$
			$6^{1}17^{1}$
5	8	**104**	$2^{100}4^{1}$
			$2^{65}4^{1}13^{1}$
			$2^{52}52^{1}$
			$2^{51}4^{1}26^{1}$
			$8^{1}13^{1}$

Parents	Designs	Runs	
3	4	**105**	$3^{1}35^{1}$
			$5^{1}21^{1}$
			$7^{1}15^{1}$
1	1	**106**	$2^{1}53^{1}$
39	79	**108**	2^{107}
			$2^{40}6^{1}$
			$2^{34}3^{29}6^{1}$
			$2^{27}3^{33}9^{1}$
			$2^{22}27^{1}$
			$2^{21}3^{1}6^{2}$
			$2^{20}3^{34}9^{1}$
			$2^{18}3^{33}6^{1}9^{1}$
			$2^{18}3^{31}18^{1}$
			$2^{17}3^{29}6^{2}$
			$2^{15}6^{1}18^{1}$
			$2^{13}3^{30}6^{1}18^{1}$
			$2^{13}6^{3}$
			$2^{12}3^{29}6^{3}$
			$2^{10}3^{40}6^{1}9^{1}$
			$2^{10}3^{33}6^{2}9^{1}$
			$2^{10}3^{31}6^{1}18^{1}$
			$2^{9}3^{36}6^{2}9^{1}$
			$2^{9}3^{34}6^{1}18^{1}$
			$2^{8}3^{30}6^{2}18^{1}$
			$2^{4}3^{33}6^{3}9^{1}$
			$2^{4}3^{31}6^{2}18^{1}$
			$2^{3}3^{41}6^{1}9^{1}$
			$2^{3}3^{39}18^{1}$
			$2^{3}3^{34}6^{3}9^{1}$
			$2^{3}3^{32}6^{2}18^{1}$
			$2^{3}3^{16}6^{8}$
			$2^{2}3^{42}18^{1}$
			$2^{2}3^{37}6^{2}9^{1}$
			$2^{2}3^{35}6^{1}18^{1}$
			$2^{2}54^{1}$
			$2^{1}3^{35}6^{3}9^{1}$
			$2^{1}3^{33}6^{2}18^{1}$
			$3^{44}9^{1}12^{1}$
			$3^{39}6^{3}9^{1}$
			$3^{37}6^{2}18^{1}$
			$3^{36}36^{1}$
			$3^{4}6^{11}$
			$4^{1}27^{1}$
3	4	**110**	$2^{1}55^{1}$
			$5^{1}22^{1}$
			$10^{1}11^{1}$
1	1	**111**	$3^{1}37^{1}$
7	57	**112**	$2^{104}8^{1}$
			$2^{89}7^{1}8^{1}$
			$2^{79}8^{1}14^{1}$
			$2^{75}4^{3}28^{1}$
			$2^{56}56^{1}$
			$4^{12}28^{1}$
			$7^{1}16^{1}$
3	4	**114**	$2^{1}57^{1}$
			$3^{1}38^{1}$
			$6^{1}19^{1}$
1	1	**115**	$5^{1}23^{1}$
4	4	**116**	2^{115}
			$2^{23}29^{1}$
			$2^{2}58^{1}$
			$4^{1}29^{1}$
2	3	**117**	$3^{13}39^{1}$
			$9^{1}13^{1}$
1	1	**118**	$2^{1}59^{1}$
1	1	**119**	$7^{1}17^{1}$
16	31	**120**	$2^{116}4^{1}$
			$2^{87}3^{1}4^{1}$
			$2^{79}4^{1}5^{1}$
			$2^{75}4^{1}6^{1}$
			$2^{75}4^{1}10^{1}$
			$2^{74}4^{1}15^{1}$
			$2^{70}3^{1}4^{1}10^{1}$
			$2^{70}4^{1}5^{1}6^{1}$
			$2^{68}4^{1}6^{1}10^{1}$
			$2^{60}60^{1}$
			$2^{59}4^{1}30^{1}$
			$2^{30}6^{1}20^{1}$
			$2^{28}10^{1}12^{1}$
			$3^{1}40^{1}$
			$5^{1}24^{1}$
			$8^{1}15^{1}$
1	1	**121**	11^{12}
1	1	**122**	$2^{1}61^{1}$
1	1	**123**	$3^{1}41^{1}$
4	4	**124**	2^{123}
			$2^{22}31^{1}$
			$2^{2}62^{1}$
			$4^{1}31^{1}$
1	2	**125**	$5^{25}25^{1}$
7	10	**126**	$2^{1}63^{1}$
			$3^{24}14^{1}$
			$3^{23}6^{1}7^{1}$
			$3^{21}42^{1}$
			$3^{20}6^{1}21^{1}$
			$7^{1}18^{1}$
			$9^{1}14^{1}$
21	740	**128**	$2^{64}64^{1}$
			$2^{6}4^{33}8^{1}16^{1}$
			$2^{6}4^{26}8^{4}16^{1}$
			$2^{6}4^{19}8^{7}16^{1}$
			$2^{6}4^{12}8^{10}16^{1}$
			$2^{6}4^{5}8^{13}16^{1}$
			$2^{5}4^{31}8^{2}16^{1}$
			$2^{5}4^{24}8^{5}16^{1}$
			$2^{5}4^{17}8^{8}16^{1}$
			$2^{5}4^{10}8^{11}16^{1}$
			$2^{5}4^{8}8^{14}$
			$2^{4}4^{36}16^{1}$
			$2^{4}4^{29}8^{3}16^{1}$
			$2^{4}4^{22}8^{6}16^{1}$
			$2^{4}4^{15}8^{9}16^{1}$
			$2^{4}4^{8}8^{12}16^{1}$
			$2^{3}4^{25}8^{7}$
			$2^{3}4^{18}8^{10}$
			$2^{3}4^{11}8^{13}$
			$4^{32}32^{1}$
			$8^{16}16^{1}$
1	1	**129**	$3^{1}43^{1}$
3	4	**130**	$2^{1}65^{1}$
			$5^{1}26^{1}$
			$10^{1}13^{1}$
11	14	**132**	2^{131}
			$2^{42}6^{1}$
			$2^{27}11^{1}$
			$2^{22}33^{1}$
			$2^{18}3^{1}22^{1}$
			$2^{18}6^{1}11^{1}$
			$2^{15}6^{1}22^{1}$
			$2^{2}66^{1}$
			$3^{1}44^{1}$
			$4^{1}33^{1}$
			$11^{1}12^{1}$
1	1	**133**	$7^{1}19^{1}$
1	1	**134**	$2^{1}67^{1}$
3	6	**135**	$3^{32}9^{1}15^{1}$
			$3^{27}45^{1}$
			$5^{1}27^{1}$
5	8	**136**	$2^{132}4^{1}$
			$2^{83}4^{1}17^{1}$
			$2^{68}68^{1}$
			$2^{67}4^{1}34^{1}$
			$8^{1}17^{1}$
3	4	**138**	$2^{1}69^{1}$
			$3^{1}46^{1}$
			$6^{1}23^{1}$

Parents	Designs	Runs	
13	15	**140**	2^{139}
			$2^{38}7^{1}$
			$2^{36}10^{1}$
			$2^{34}14^{1}$
			$2^{27}5^{1}7^{1}$
			$2^{25}5^{1}14^{1}$
			$2^{22}35^{1}$
			$2^{21}7^{1}10^{1}$
			$2^{17}10^{1}14^{1}$
			$2^{2}70^{1}$
			$4^{1}35^{1}$
			$5^{1}28^{1}$
			$7^{1}20^{1}$
1	1	**141**	$3^{1}47^{1}$
1	1	**142**	$2^{1}71^{1}$
1	1	**143**	$11^{1}13^{1}$
16	1,241	**144**	$2^{136}8^{1}$
			$2^{117}8^{1}9^{1}$
			$2^{113}3^{1}24^{1}$
			$2^{111}6^{1}24^{1}$
			$2^{103}8^{1}18^{1}$
			$2^{76}3^{12}6^{4}8^{1}$
			$2^{76}3^{7}4^{1}6^{5}12^{1}$
			$2^{75}3^{3}4^{1}6^{6}12^{1}$
			$2^{74}3^{4}6^{6}8^{1}$
			$2^{72}72^{1}$
			$2^{44}3^{11}12^{2}$
			$2^{16}3^{3}6^{6}24^{1}$
			$3^{48}48^{1}$
			$4^{36}36^{1}$
			$4^{11}12^{2}$
			12^{7}
1	2	**147**	$7^{9}21^{1}$
1	1	**148**	2^{147}
1	5	**150**	$5^{11}30^{1}$
2	6	**152**	$2^{148}4^{1}$
			$2^{76}76^{1}$
1	1	**153**	$3^{25}17^{1}$
1	1	**156**	2^{155}
6	110	**160**	$2^{144}16^{1}$
			$2^{138}4^{7}$
			$2^{133}5^{1}16^{1}$
			$2^{127}10^{1}16^{1}$
			$2^{80}80^{1}$
			$4^{16}40^{1}$
3	61	**162**	$3^{65}6^{1}27^{1}$
			$3^{54}54^{1}$
			$9^{18}18^{1}$
1	1	**164**	2^{163}
2	16	**168**	$2^{164}4^{1}$
			$2^{84}84^{1}$
1	1	**169**	13^{14}
1	1	**171**	$3^{28}19^{1}$
1	1	**172**	2^{171}
1	2	**175**	$5^{10}35^{1}$
4	56	**176**	$2^{168}8^{1}$
			$2^{166}4^{3}$
			$2^{88}88^{1}$
			$4^{12}44^{1}$
3	24	**180**	2^{179}
			$3^{30}60^{1}$
			$6^{2}30^{1}$
2	6	**184**	$2^{180}4^{1}$
			$2^{92}92^{1}$
1	1	**188**	2^{187}
1	4	**189**	$3^{36}63^{1}$
4	726	**192**	$2^{160}32^{1}$
			$2^{96}96^{1}$
			$4^{48}48^{1}$
			$8^{8}24^{1}$
3	11	**196**	2^{195}
			$7^{28}28^{1}$
			14^{5}
1	5	**198**	$3^{30}66^{1}$
4	42	**200**	$2^{196}4^{1}$
			$2^{100}100^{1}$
			$5^{20}40^{1}$
			$10^{5}20^{1}$
1	1	**204**	2^{203}
1	1	**207**	$3^{25}23^{1}$
4	72	**208**	$2^{200}8^{1}$
			$2^{198}4^{3}$
			$2^{104}104^{1}$
			$4^{16}52^{1}$
1	1	**212**	2^{211}
6	258	**216**	$2^{212}4^{1}$
			$2^{108}108^{1}$
			$2^{11}3^{77}12^{1}18^{1}$
			$3^{72}72^{1}$
			$3^{66}6^{5}12^{1}18^{1}$
			$6^{7}36^{1}$
1	1	**220**	2^{219}
5	351	**224**	$2^{208}16^{1}$
			$2^{193}7^{1}16^{1}$
			$2^{183}14^{1}16^{1}$
			$2^{112}112^{1}$
			$4^{56}56^{1}$
3	13	**225**	$3^{27}75^{1}$
			$5^{20}45^{1}$
			15^{6}
1	1	**228**	2^{227}
2	6	**232**	$2^{228}4^{1}$
			$2^{116}116^{1}$
1	5	**234**	$3^{30}78^{1}$
1	1	**236**	2^{235}
6	302	**240**	$2^{232}8^{1}$
			$2^{230}4^{3}$
			$2^{205}5^{1}24^{1}$
			$2^{199}10^{1}24^{1}$
			$2^{120}120^{1}$
			$4^{20}60^{1}$
1	2	**242**	$11^{22}22^{1}$
2	58	**243**	$3^{81}81^{1}$
			$9^{27}27^{1}$
1	1	**244**	2^{243}
1	2	**245**	$7^{10}35^{1}$
2	6	**248**	$2^{244}4^{1}$
			$2^{124}124^{1}$
1	4	**250**	$5^{50}50^{1}$
3	23	**252**	2^{251}
			$3^{42}84^{1}$
			$6^{2}42^{1}$
2	6,101	**256**	$8^{32}32^{1}$
			16^{17}
1	1	**260**	2^{259}
1	2	**261**	$3^{27}87^{1}$
2	16	**264**	$2^{260}4^{1}$
			$2^{132}132^{1}$
1	1	**268**	2^{267}
1	11	**270**	$3^{90}90^{1}$
4	136	**272**	$2^{264}8^{1}$
			$2^{262}4^{3}$
			$2^{136}136^{1}$
			$4^{32}68^{1}$
1	2	**275**	$5^{11}55^{1}$
1	1	**276**	2^{275}
1	2	**279**	$3^{30}93^{1}$
2	17	**280**	$2^{276}4^{1}$
			$2^{140}140^{1}$
1	1	**284**	2^{283}

Parents	Designs	Runs	
8	3,201	**288**	$2^{272}16^{1}$
			$2^{253}9^{1}16^{1}$
			$2^{239}16^{1}18^{1}$
			$2^{144}144^{1}$
			$3^{96}96^{1}$
			$4^{36}72^{1}$
			$6^{10}48^{1}$
			$12^{6}24^{1}$
1	1	**289**	17^{18}
1	1	**292**	2^{291}
1	5	**294**	$7^{18}42^{1}$
1	2	**296**	$2^{292}4^{1}$
1	4	**297**	$3^{39}99^{1}$
3	24	**300**	2^{299}
			$5^{20}60^{1}$
			$10^{2}30^{1}$
4	70	**304**	$2^{296}8^{1}$
			$2^{294}4^{3}$
			$2^{228}76^{1}$
			$4^{16}76^{1}$
1	5	**306**	$3^{48}102^{1}$
1	1	**308**	2^{307}
1	2	**312**	$2^{308}4^{1}$
1	5	**315**	$3^{29}105^{1}$
1	1	**316**	2^{315}
5	725	**320**	$2^{288}32^{1}$
			$2^{274}4^{15}$
			$2^{240}80^{1}$
			$4^{40}80^{1}$
			$8^{10}40^{1}$
6	974	**324**	2^{323}
			$3^{143}12^{1}27^{1}$
			$3^{108}108^{1}$
			$6^{2}54^{1}$
			$9^{36}36^{1}$
			18^{3}
1	2	**325**	$5^{20}65^{1}$
1	2	**328**	$2^{324}4^{1}$
1	1	**332**	2^{331}
1	2	**333**	$3^{36}111^{1}$
6	487	**336**	$2^{328}8^{1}$
			$2^{326}4^{3}$
			$2^{297}7^{1}24^{1}$
			$2^{287}14^{1}24^{1}$
			$2^{252}84^{1}$
			$4^{36}84^{1}$
1	2	**338**	$13^{26}26^{1}$
1	1	**340**	2^{339}
1	5	**342**	$3^{30}114^{1}$
1	2	**343**	$7^{49}49^{1}$
1	2	**344**	$2^{340}4^{1}$
1	1	**348**	2^{347}
1	5	**350**	$5^{20}70^{1}$
1	4	**351**	$3^{39}117^{1}$
4	203	**352**	$2^{336}16^{1}$
			$2^{330}4^{7}$
			$2^{264}88^{1}$
			$4^{32}88^{1}$
1	1	**356**	2^{355}
3	133	**360**	$2^{356}4^{1}$
			$3^{48}120^{1}$
			$6^{8}60^{1}$
1	1	**361**	19^{20}
1	2	**363**	$11^{11}33^{1}$
1	1	**364**	2^{363}
4	150	**368**	$2^{360}8^{1}$
			$2^{358}4^{3}$
			$2^{276}92^{1}$
			$4^{36}92^{1}$
1	2	**369**	$3^{30}123^{1}$
1	1	**372**	2^{371}
1	4	**375**	$5^{40}75^{1}$
1	2	**376**	$2^{372}4^{1}$
1	11	**378**	$3^{72}126^{1}$
1	1	**380**	2^{379}
3	3,252	**384**	$2^{320}64^{1}$
			$4^{96}96^{1}$
			$8^{16}48^{1}$
1	2	**387**	$3^{48}129^{1}$
1	1	**388**	2^{387}
4	28	**392**	$2^{388}4^{1}$
			$2^{196}7^{28}28^{1}$
			$7^{28}56^{1}$
			$14^{5}28^{1}$
3	23	**396**	2^{395}
			$3^{132}132^{1}$
			$6^{2}66^{1}$
7	912	**400**	$2^{392}8^{1}$
			$2^{390}4^{3}$
			$2^{300}100^{1}$
			$4^{36}100^{1}$
			$5^{80}80^{1}$
			$10^{6}40^{1}$
			20^{5}
1	1	**404**	2^{403}
2	60	**405**	$3^{81}135^{1}$
			$9^{18}45^{1}$
1	2	**408**	$2^{404}4^{1}$
1	1	**412**	2^{411}
1	5	**414**	$3^{48}138^{1}$
4	299	**416**	$2^{400}16^{1}$
			$2^{394}4^{7}$
			$2^{312}104^{1}$
			$4^{48}104^{1}$
1	1	**420**	2^{419}
1	2	**423**	$3^{30}141^{1}$
1	2	**424**	$2^{420}4^{1}$
1	2	**425**	$5^{20}85^{1}$
1	1	**428**	2^{427}
7	10,839	**432**	$2^{424}8^{1}$
			$2^{389}9^{1}24^{1}$
			$2^{375}18^{1}24^{1}$
			$3^{144}144^{1}$
			$4^{108}108^{1}$
			$6^{12}72^{1}$
			$12^{6}36^{1}$
1	1	**436**	2^{435}
1	2	**440**	$2^{436}4^{1}$
3	12	**441**	$3^{42}7^{7}21^{1}$
			$7^{14}63^{1}$
			21^{7}
1	1	**444**	2^{443}
4	8,598	**448**	$2^{416}32^{1}$
			$2^{336}112^{1}$
			$4^{56}112^{1}$
			$8^{56}56^{1}$
3	35	**450**	$3^{150}5^{11}30^{1}$
			$5^{90}90^{1}$
			$15^{5}30^{1}$
1	2	**456**	$2^{452}4^{1}$
1	2	**459**	$3^{72}9^{1}17^{1}$
1	1	**460**	2^{459}
4	150	**464**	$2^{456}8^{1}$
			$2^{454}4^{3}$
			$2^{348}116^{1}$
			$4^{36}116^{1}$
3	17	**468**	2^{467}
			$3^{49}52^{1}$
			$6^{2}78^{1}$
1	2	**472**	$2^{468}4^{1}$
1	2	**475**	$5^{20}95^{1}$
1	1	**477**	$3^{37}53^{1}$
4	1,210	**480**	$2^{464}16^{1}$
			$2^{458}4^{7}$
			$2^{360}120^{1}$
			$4^{56}120^{1}$

Parents	Designs	Runs	
3	8	**484**	2^{483}
			$11^{44}44^{1}$
			22^{3}
1	277	**486**	$9^{54}54^{1}$
1	2	**488**	$2^{484}4^{1}$
1	5	**490**	$7^{18}70^{1}$
1	1	**492**	2^{491}
1	2	**495**	$3^{42}5^{1}33^{1}$
4	78	**496**	$2^{488}8^{1}$
			$2^{486}4^{3}$
			$2^{372}124^{1}$
			$4^{18}124^{1}$
3	29	**500**	2^{499}
			$5^{100}100^{1}$
			$10^{2}50^{1}$
3	111	**504**	$2^{500}4^{1}$
			$2^{84}3^{84}84^{1}$
			$6^{8}84^{1}$
2	73,992	**512**	$8^{64}64^{1}$
			$16^{32}32^{1}$
1	2	**513**	$3^{81}9^{1}19^{1}$
733	**117,561**		

The following step provides a simple example of using the `%MktEx` macro to request the L_{36} design, $2^{11}3^{12}$, which has 11 two-level factors and 12 three-level factors:

```
%mktex(n=36)
```

No iterations are needed, and the macro immediately creates the L_{36}, which is 100% efficient. This example runs in a few seconds. The factors are always named `x1`, `x2`, ... and the levels are always consecutive integers starting with 1. You can use the `%MktLab` macro to assign different names and levels (see page 1093).

Randomization

By default, the macro creates two output data sets with the design—one sorted and one randomized

- `out=Design` – the experimental design, sorted by the factor levels.
- `outr=Randomized` – the randomized experimental design.

The two designs are equivalent and have the same *D*-efficiency. The `out=Design` data set is sorted and hence is usually easier to look at, however the `outr=Randomized` design is usually the better one to use. The randomized design has the rows sorted into a random order, and all of the factor levels are randomly reassigned. For example with two-level factors, approximately half of the original (1, 2) mappings are reassigned (2, 1). Similarly, with three level factors, the mapping (1, 2, 3) are changed to one of the following: (1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), or (3, 2, 1). The reassignment of levels is usually not critical for the iteratively derived designs, but it can be very important the orthogonal designs, many of which have all ones in the first row.

Latin Squares and Graeco-Latin Square Designs

The `%MktEx` orthogonal array catalog can be used to make both Latin Square and Graeco-Latin Square (mutually orthogonal Latin Square) designs. A Latin square is an $p \times p$ table with p different values arranged so that each value occurs exactly once in each row and exactly once in each column. An orthogonal array p^3 in p^2 runs can be used to make a Latin square. The following matrices are Latin Squares of order $p = 3, 3, 4, 5, 6$:

$$\begin{bmatrix} 1 & 3 & 2 \\ 3 & 2 & 1 \\ 2 & 1 & 3 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \\ 2 & 3 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \\ 3 & 4 & 1 & 2 \\ 4 & 3 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 3 & 5 & 2 & 4 \\ 5 & 2 & 4 & 1 & 3 \\ 4 & 1 & 3 & 5 & 2 \\ 3 & 5 & 2 & 4 & 1 \\ 2 & 4 & 1 & 3 & 5 \end{bmatrix} \quad \begin{bmatrix} 3 & 6 & 4 & 1 & 5 & 2 \\ 1 & 4 & 5 & 2 & 6 & 3 \\ 2 & 5 & 3 & 6 & 1 & 4 \\ 4 & 1 & 2 & 5 & 3 & 6 \\ 5 & 2 & 6 & 3 & 4 & 1 \\ 6 & 3 & 1 & 4 & 2 & 5 \end{bmatrix}$$

The first two Latin Squares are obtained from `%MktEx` as follows:

```
%mktex(3 ** 4, n=3 * 3)

proc print; run;
```

The design is as follows:

```
Obs    x1    x2    x3    x4

 1      1     1     1     1
 2      1     2     3     2
 3      1     3     2     3
 4      2     1     3     3
 5      2     2     2     1
 6      2     3     1     2
 7      3     1     2     2
 8      3     2     1     3
 9      3     3     3     1
```

To make a Latin square from an orthogonal array, treat `x1` as the row number and `x2` as the column number. The values in `x3` form one Latin square (the first in the list shown previously), and the values in `x4` form a different Latin square (the second in the list). You can use the following macro to display the design that `%MktEx` creates in the Latin square format:

```
%macro latin(x,y);
   proc iml;
      use design;
      read all into x;
      file print;
      s1 = '123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ+=-*';
      s2 = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz123456789+=-*';
      k = 0;
      m = x[nrow(x),1];
      do i = 1 to m;
         put;
         do j = 1 to m;
            k = k + 1;
            put (substr(s1, x[k,&x], 1)) $1. @;
            %if &y ne %then %do; put +1 (substr(s2, x[k,&y], 1)) $1. +1 @; %end;
            put +1 @;
            end;
         end;

      put;
      quit;
   %mend;
```

You specify the column number as the first parameter of the macro. You can specify any *p*-level factor after the first two. The following statements create and display the five Latin squares that are displayed previously:

```
%mktex(3 ** 4, n=3 * 3)
%latin(3)
%latin(4)

%mktex(4 ** 5, n=4 ** 2)
%latin(3)

%mktex(5 ** 3, n=25)
%latin(3)

%mktex(6 ** 3, n=36)
%latin(3)
```

Note that you can specify a number or an expression for `n=`, and this example does both. The results (from the `latin` macro only) are as follows:

```
1 3 2
3 2 1
2 1 3

1 2 3
3 1 2
2 3 1

1 2 3 4
2 1 4 3
3 4 1 2
4 3 2 1

1 3 5 2 4
5 2 4 1 3
4 1 3 5 2
3 5 2 4 1
2 4 1 3 5

3 6 4 1 5 2
1 4 5 2 6 3
2 5 3 6 1 4
4 1 2 5 3 6
5 2 6 3 4 1
6 3 1 4 2 5
```

Alternatively, you can construct a Latin square from the randomized design, for example, as follows:

```
%mktex(6 ** 3,                    /* 3 six-level factor                    */
       n=36,                      /* 36 runs                               */
       options=nohistory          /* do not display iteration history      */
               nofinal,           /* do not display final levels, D-eff    */
       seed=109)                  /* random number seed                    */

proc sort data=randomized out=design; by x1 x2; run;

%latin(3)
```

With different random number seeds, you will typically get different Latin squares, particularly for larger Latin squares. The results of this step are as follows:

```
4 3 1 6 5 2
2 5 6 3 4 1
1 6 3 5 2 4
5 4 2 1 3 6
6 2 5 4 1 3
3 1 4 2 6 5
```

When an orthogonal array has 3 or more p-level factors in p^2 runs, you can make one or more Latin squares. When an orthogonal array has 4 or more p-level factors in p^2 runs, you can make one or more Graeco-Latin squares, which are also known as mutually orthogonal Latin squares or Euler squares (named after the Swiss mathematician Leonhard Euler). The following example creates and displays a Graeco-Latin square of order $p = 3$:

```
%mktex(3 ** 4, n=3 * 3)
%latin(3,4)
```

The results are as follows:

```
1 A  3 B  2 C
3 C  2 A  1 B
2 B  1 C  3 A
```

Each entry consists of two values (the two columns of the design that are specified in the `latin` macro). The $p \times p = 9$ left-most values form a Latin square as do the $p \times p$ right-most values. In addition, the two factors are orthogonal to each other and to the row and column indexes (`x1` and `x2`). Graeco-Latin squares exist for all $p \geq 3$ except $p = 6$. The following steps create and display a Graeco-Latin square of order $p = 12$:

```
%mktex(12 ** 4, n=12 ** 2)
%latin(3,4)
```

The results are as follows:

```
1 A  7 G  5 H  b I  3 F  9 K  8 L  6 C  c E  4 J  a D  2 B
a K  2 A  8 G  6 H  c I  4 F  3 B  9 L  1 C  7 E  5 J  b D
5 F  b K  3 A  9 G  1 H  7 I  c D  4 B  a L  2 C  8 E  6 J
8 I  6 F  c K  4 A  a G  2 H  1 J  7 D  5 B  b L  3 C  9 E
3 H  9 I  1 F  7 K  5 A  b G  a E  2 J  8 D  6 B  c L  4 C
c G  4 H  a I  2 F  8 K  6 A  5 C  b E  3 J  9 D  1 B  7 L
2 L  c C  6 E  a J  4 D  8 B  7 A  1 G  b H  5 I  9 F  3 K
9 B  3 L  7 C  1 E  b J  5 D  4 K  8 A  2 G  c H  6 I  a F
6 D  a B  4 L  8 C  2 E  c J  b F  5 K  9 A  3 G  7 H  1 I
7 J  1 D  b B  5 L  9 C  3 E  2 I  c F  6 K  a A  4 G  8 H
4 E  8 J  2 D  c B  6 L  a C  9 H  3 I  7 F  1 K  b A  5 G
b C  5 E  9 J  3 D  7 B  1 L  6 G  a H  4 I  8 F  2 K  c A
```

The `latin` macro uses numerals, lower-case letters, upper case letters, and symbols to display the first Latin square. It uses upper case letters, lower-case letters, numerals, and symbols to display the second Latin square. Up to 64 different values can be displayed. Currently, the `%MktEx` macro is capable of making Graeco-Latin squares for all orders in the range 3–25 except 6 (does not exist), 18, and 22. It can make a few larger ones as well (when p is a power of a prime like 49, 64, and 81).

Split-Plot Designs

In a split-plot experiment, there are blocks or plots, and some factors can only be applied to an entire plot. The term "plot" comes from agricultural experiments, where it is only convenient to apply some treatments to entire plots of land. These factors are called "whole-plot factors." Within a plot, other factors can be applied to smaller sections, and these are called "subplot factors." The goal is to create an experimental design with n rows, and p plots of size s where $n = ps$. All whole plot factors must be constant within each of the p plots of size s.

Split-Plot Designs from Orthogonal Arrays

The following split-plot design, with 4 plots of size 6, is constructed from the orthogonal array 2^{23} in 24 runs:

```
 Plot  Whole   ------------Subplot Factors------------

    1  1 1 1   2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       1 1 1   2 2 1 1 1 2 1 1 2 1 2 2 1 1 1 2 1 1 2 1
       1 1 1   1 1 2 1 2 2 2 1 1 1 1 1 2 1 2 2 2 1 1 1
       1 1 1   2 1 1 1 2 1 1 2 1 2 2 1 1 1 2 1 1 2 1 2
       1 1 1   1 1 1 2 1 1 2 1 2 2 1 1 1 2 1 1 2 1 2 2
       1 1 1   1 2 2 2 1 1 1 2 1 1 1 2 2 2 1 1 1 2 1 1

    2  1 2 2   1 2 1 2 2 2 1 1 1 2 1 2 1 2 2 2 1 1 1 2
       1 2 2   2 1 2 2 2 1 1 1 2 1 2 1 2 2 2 1 1 1 2 1
       1 2 2   1 1 2 1 1 2 1 2 2 2 1 1 2 1 1 2 1 2 2 2
       1 2 2   2 1 1 2 1 2 2 2 1 1 2 1 1 2 1 2 2 2 1 1
       1 2 2   1 2 1 1 2 1 2 2 2 1 1 2 1 1 2 1 2 2 2 1
       1 2 2   2 2 2 1 1 1 2 1 1 2 2 2 2 1 1 1 2 1 1 2

    3  2 2 1   2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
       2 2 1   2 2 1 1 1 2 1 1 2 1 1 1 2 2 2 1 2 2 1 2
       2 2 1   1 1 2 1 2 2 2 1 1 1 2 2 1 2 1 1 1 2 2 2
       2 2 1   2 1 1 1 2 1 1 2 1 2 1 2 2 2 1 2 2 1 2 1
       2 2 1   1 1 1 2 1 1 2 1 2 2 2 2 2 1 2 2 1 2 1 1
       2 2 1   1 2 2 2 1 1 1 2 1 1 2 1 1 1 2 2 2 1 2 2
```

```
     4  2 1 2   1 2 1 2 2 2 1 1 1 2 2 1 2 1 1 1 2 2 2 1
        2 1 2   2 1 2 2 2 1 1 1 2 1 1 2 1 1 1 2 2 2 1 2
        2 1 2   1 1 2 1 1 2 1 2 2 2 2 2 1 2 2 1 2 1 1 1
        2 1 2   2 1 1 2 1 2 2 2 1 1 1 2 2 1 2 1 1 1 2 2
        2 1 2   1 2 1 1 2 1 2 2 2 1 2 1 2 2 1 2 1 1 1 2
        2 1 2   2 2 2 1 1 1 2 1 1 2 1 1 1 2 2 2 1 2 2 1
```

Notice that there is a maximum of three whole-plot factors available in this design, and that all of the 21 subplot-factors are perfectly balanced within each plot. You can construct this design as follows:

```
%mktex(2 ** 23, n=24, options=nosort)

data splitplot;
   Plot = floor((_n_ - 1) / 6) + 1; /* 6 1s, 6 2s, 6 3s, 6 4s */
   set design;
   run;

proc print; id Plot; by Plot; var x22 x23 x21 x1-x20; run;
```

The `nosort` option is used to output the design just as it is created, without sorting first. In this particular case, it leads to a particularly nice arrangement of the design where the whole-plot factors can be constructed from `x22`, `x23`, and `x21`. To understand why this is the case, you need to see the lineage or construction method for this particular design. The design lineage is a set of instructions for making a design with smaller-level factors from a design with higher-level factors. You can begin by adding the `lineage` option as follows:

```
%mktex(2 ** 23, n=24, options=nosort lineage)
```

In addition to making the design, the `%MktEx` macro reports the following:

```
Design Lineage:
24 ** 1 : 24 ** 1 > 2 ** 20 4 ** 1 : 4 ** 1 > 2 ** 3
```

The first part of the lineage states that the design begins as a single 24-level factor,[*] and it is replaced by the orthogonal array $2^{20}4^1$. The final array is constructed from $2^{20}4^1$ by replacing the four-level factor with 3 two-level factors (4^1 with 2^3), which creates the 3 two-level factors that we use for our whole-plot factors. Note that the factors come out in the order described by the lineage: the 20 two-level factors first, then the 3 two-level factors that come from the four-level factor. You cannot get the two-level factors that come from the four-level factor without also requesting the other 20 two-level factors. We want the two-level factors that come from the four-level factor for our whole-plot factors, because in an orthogonal array, the underlying four-level factor must be orthogonal to a six-level factor. This ensures that each of the four combinations of these particular two-level factors occurs exactly six times in the final design. This is not guaranteed for other triples of two-level factors in the design.

Now consider selecting whole-plot factors from other two-level factors, for example as follows:

[*]This allows the same code that replaces a 24-level factor (for example in 48, 72, 96, 120, or 144 runs) to be used to make designs in 24 runs.

```
proc sort data=design out=splitplot;
   by x1-x23;
   run;

data splitplot;
   Plot = floor((_n_ - 1) / 6) + 1;
   set splitplot;
   run;

proc print; id Plot; by Plot; run;
```

The results are as follows:

```
P
l                             x  x  x  x  x  x  x  x  x  x  x  x  x  x
o  x  x  x  x  x  x  x  x  x  1  1  1  1  1  1  1  1  1  1  2  2  2  2
t  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5  6  7  8  9  0  1  2  3

1  1  1  1  2  1  1  2  1  2  2  1  1  1  2  1  1  2  1  2  2  1  1  1
   1  1  1  2  1  1  2  1  2  2  2  2  2  1  2  2  1  2  1  1  1  2  2
   1  1  2  1  1  2  1  2  2  2  1  1  2  1  1  2  1  2  2  2  2  1  2
   1  1  2  1  1  2  1  2  2  2  2  2  1  2  2  1  2  1  1  1  2  2  1
   1  1  2  1  2  2  2  1  1  1  1  1  2  1  2  2  2  1  1  1  1  1  1
   1  1  2  1  2  2  2  1  1  1  2  2  1  2  1  1  1  2  2  2  1  2  2

2  1  2  1  1  2  1  2  2  2  1  1  2  1  1  2  1  2  2  2  1  2  1  2
   1  2  1  1  2  1  2  2  2  1  2  1  2  2  1  2  1  1  1  2  2  2  1
   1  2  1  2  2  2  1  1  1  2  1  2  1  2  2  2  1  1  1  2  2  1  2
   1  2  1  2  2  2  1  1  1  2  2  1  2  1  1  1  2  2  2  1  2  2  1
   1  2  2  2  1  1  1  2  1  1  1  2  2  2  1  1  1  2  1  1  1  1  1
   1  2  2  2  1  1  1  2  1  1  2  1  1  1  2  2  2  1  2  2  1  2  2

3  2  1  1  1  2  1  1  2  1  2  1  2  2  2  1  2  2  1  2  1  1  2  2
   2  1  1  1  2  1  1  2  1  2  2  1  1  1  2  1  1  2  1  2  1  1  1
   2  1  1  2  1  2  2  2  1  1  1  2  2  1  2  1  1  1  2  2  2  2  1
   2  1  1  2  1  2  2  2  1  1  2  1  1  2  1  2  2  2  1  1  2  1  2
   2  1  2  2  2  1  1  1  2  1  1  2  1  1  1  2  2  2  1  2  2  2  1
   2  1  2  2  2  1  1  1  2  1  2  1  2  2  2  1  1  1  2  1  2  1  2

4  2  2  1  1  1  2  1  1  2  1  1  1  2  2  2  1  2  2  1  2  1  2  2
   2  2  1  1  1  2  1  1  2  1  2  2  1  1  1  2  1  1  2  1  1  1  1
   2  2  2  1  1  1  2  1  1  2  1  1  1  2  2  2  1  2  2  1  2  2  1
   2  2  2  1  1  1  2  1  1  2  2  2  2  1  1  1  2  1  1  2  2  1  2
   2  2  2  2  2  2  2  2  2  2  1  1  1  1  1  1  1  1  1  1  1  2  2
   2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  1  1  1
```

Now there are only two whole-plot factors available, and many of the subplot factors are not balanced within each plot. Many split-plot designs can be constructed by carefully selecting factors from an orthogonal array. To take full advantage of the orthogonal array capabilities in the `%MktEx` macro for use in split-plot designs, you need to understand something else about design lineages. Some designs, such as 2^{23}, can be constructed in multiple ways. Some of those ways work better for our purposes than others. Consider the following steps:

```
%mktorth(range=n=24, options=dups lineage)

data cat;
   set mktdescat;
   design  = left(transtrn(compbl(design), ' ** ', '^'));
   lineage = left(transtrn(substr(lineage, 21), ' ** ', '^'));
   run;

proc print noobs; run;
```

The `%MktOrth` macro manages the catalog of orthogonal arrays that the `%MktEx` macro can make, and provides the `%MktEx` macro with instructions (when the `lineage` option is specified) about how the designs are made. The option `range=n=24` outputs instructions for only designs in 24 runs. The `dups` option suppresses normal filtering and removal of duplicate and inferior designs. A design is considered to be a duplicate of another design if it has exactly the same numbers of each type of factor as some other design. An inferior design has only a subset of the factors that are available in a competing design. This determination is solely based on the number of each type of factor, not on the actual levels of the factors. Depending on your purposes, a duplicate or inferior design might be a better than the design that `%MktEx` makes by default, because different construction methods lead to designs with different combinatorial properties. For this reason, `%MktEx` provides you with a way to create the designs that it filters out by default.

The `%MktOrth` macro creates two data sets. The MKTDESCAT data set is smaller and more suitable for display. The MKTDESLEV dat set is larger, and has more information for processing and subsetting. In particular, the MKTDESLEV data set has a variable `x2` that contains the number of two-level factors, `x3` that contains the number of three-level factors, and so on. The `%MktEx` macro uses the information in the MKTDESLEV dat set when it creates orthogonal arrays.

The DATA step simply subsets and reformats the results into a form that can be displayed in one panel without splitting. The `transtrn` function replaces the string ' ** ' with '^' indicating an exponent in much less space. Additionally, the parts of the lineages dealing with 24 level factors being replaced by a parent along with extra blanks are removed. The results are displayed next:

```
  n  Design        Reference         Lineage

 24  2^23          Hadamard          2^12 12^1 : 12^1 > 2^11
 24  2^23          Hadamard          2^20 4^1 : 4^1 > 2^3
 24  2^20 4^1      Orthogonal Array  2^20 4^1 (parent)
 24  2^16 3^1      Orthogonal Array  2^12 12^1 : 12^1 > 2^4 3^1
 24  2^16 3^1      Orthogonal Array  2^13 3^1 4^1 : 4^1 > 2^3
 24  2^15 3^1      Orthogonal Array  2^11 4^1 6^1 : 6^1 > 2^1 3^1 : 4^1 > 2^3
 24  2^15 3^1      Orthogonal Array  2^12 12^1 : 12^1 > 2^2 6^1 : 6^1 > 2^1 3^1
 24  2^15 3^1      Orthogonal Array  2^12 12^1 : 12^1 > 3^1 4^1 : 4^1 > 2^3
 24  2^14 6^1      Orthogonal Array  2^11 4^1 6^1 : 4^1 > 2^3
 24  2^14 6^1      Orthogonal Array  2^12 12^1 : 12^1 > 2^2 6^1
 24  2^13 3^1 4^1  Orthogonal Array  2^13 3^1 4^1 (parent)
 24  2^12 3^1 4^1  Orthogonal Array  2^11 4^1 6^1 : 6^1 > 2^1 3^1
 24  2^12 3^1 4^1  Orthogonal Array  2^12 12^1 : 12^1 > 3^1 4^1
 24  2^12 12^1     Orthogonal Array  2^12 12^1 (parent)
 24  2^11 4^1 6^1  Orthogonal Array  2^11 4^1 6^1 (parent)
 24  2^7 3^1       Orthogonal Array  3^1 8^1 : 8^1 > 2^4 4^1 : 4^1 > 2^3
 24  2^4 3^1 4^1   Orthogonal Array  3^1 8^1 : 8^1 > 2^4 4^1
 24  3^1 8^1       Full-Factorial    3^1 8^1 (parent)
```

The design 2^{23} can be made from the two parent arrays $2^{20}4^1$ and $2^{12}12^1$. By default, the parent $2^{20}4^1$ is used, and you can see this by examining the default catalog with the default filtering of duplicate and inferior designs, for example as follows:

```
%mktorth(range=n=24, options=lineage)

data cat;
   set mktdescat;
   design  = left(transtrn(compbl(design), ' ** ', '^'));
   lineage = left(transtrn(substr(lineage, 21), ' ** ', '^'));
   run;

proc print noobs; run;
```

The results are as follows:

```
  n    Design          Reference          Lineage

 24    2^23            Hadamard           2^20 4^1 : 4^1 > 2^3
 24    2^20 4^1        Orthogonal Array   2^20 4^1 (parent)
 24    2^16 3^1        Orthogonal Array   2^13 3^1 4^1 : 4^1 > 2^3
 24    2^14 6^1        Orthogonal Array   2^11 4^1 6^1 : 4^1 > 2^3
 24    2^13 3^1 4^1    Orthogonal Array   2^13 3^1 4^1 (parent)
 24    2^12 12^1       Orthogonal Array   2^12 12^1 (parent)
 24    2^11 4^1 6^1    Orthogonal Array   2^11 4^1 6^1 (parent)
 24    3^1 8^1         Full-Factorial     3^1 8^1 (parent)
```

You can force `%MktEx` to construct the array from the parent $2^{12}12^1$ by explicitly providing `%MktEx` with a design catalog that contains the lineage of just the design of interest, for example as follows:

```
data lev;
   set mktdeslev;
   if x2 eq 23 and index(lineage, '2 ** 11');
   run;

%mktex(2 ** 23, n=24, options=nosort, cat=lev)

data splitplot;
   Plot = floor((_n_ - 1) / 6) + 1;
   set design;
   run;

proc print; id Plot; by Plot; run;
```

By default, when you do not specify the `cat=` option, `%MktEx` uses `%MktOrth` to generate the filtered catalog automatically. When you specify the `cat=` option, `%MktEx` does not use `%MktOrth` to generate the catalog, and it uses only the catalog that you provide.

The results are as follows:

```
P
l                                x  x  x  x  x  x  x  x  x  x  x  x  x  x
o  x  x  x  x  x  x  x  x  x  1  1  1  1  1  1  1  1  1  1  2  2  2  2
t  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5  6  7  8  9  0  1  2  3

1  1  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
   1  1  2  1  2  2  2  1  1  1  2  1  1  2  1  2  2  2  1  1  1  2  1
   1  1  1  2  1  2  2  2  1  1  1  2  1  1  2  1  2  2  2  1  1  1  2
   1  2  1  1  2  1  2  2  2  1  1  1  2  1  1  2  1  2  2  2  1  1  1
   1  1  2  1  1  2  1  2  2  2  1  1  1  2  1  1  2  1  2  2  2  1  1
   1  1  1  2  1  1  2  1  2  2  2  1  1  1  2  1  1  2  1  2  2  2  1
```

```
2  1  1  1  1  2  1  1  2  1  2  2  2  1  1  1  2  1  1  2  1  2  2  2
   1  2  1  1  1  2  1  1  2  1  2  2  2  1  1  1  2  1  1  2  1  2  2
   1  2  2  1  1  1  2  1  1  2  1  2  2  2  1  1  1  2  1  1  2  1  2
   1  2  2  2  1  1  1  2  1  1  2  1  2  2  2  1  1  1  2  1  1  2  1
   1  1  2  2  2  1  1  1  2  1  1  2  1  2  2  2  1  1  1  2  1  1  2
   1  2  1  2  2  2  1  1  1  2  1  1  2  1  2  2  2  1  1  1  2  1  1

3  2  1  1  1  1  1  1  1  1  1  1  1  2  2  2  2  2  2  2  2  2  2  2
   2  2  1  2  1  1  1  2  2  2  1  2  1  2  1  2  2  2  1  1  1  2  1
   2  2  2  1  2  1  1  1  2  2  2  1  1  1  2  1  2  2  2  1  1  1  2
   2  1  2  2  1  2  1  1  1  2  2  2  2  1  1  2  1  2  2  2  1  1  1
   2  2  1  2  2  1  2  1  1  1  2  2  1  2  1  1  2  1  2  2  2  1  1
   2  2  2  1  2  2  1  2  1  1  1  2  1  1  2  1  1  2  1  2  2  2  1

4  2  2  2  2  1  2  2  1  2  1  1  1  1  1  1  2  1  1  2  1  2  2  2
   2  1  2  2  2  1  2  2  1  2  1  1  2  1  1  1  2  1  1  2  1  2  2
   2  1  1  2  2  2  1  2  2  1  2  1  2  2  1  1  1  2  1  1  2  1  2
   2  1  1  1  2  2  2  1  2  2  1  2  2  2  2  1  1  1  2  1  1  2  1
   2  2  1  1  1  2  2  2  1  2  2  1  1  2  2  2  1  1  1  2  1  1  2
   2  1  2  1  1  1  2  2  2  1  2  2  2  1  2  2  2  1  1  1  2  1  1
```

Now there is only one whole-plot factor that is easily identified. Other sorts might reveal more whole-plot factors. For this problem, we would stick with the design that `%MktEx` makes by default. For other problems, you might want to force `%MktEx` to use a lineage that would otherwise be filtered out.

Split-Plot Designs from Computerized Search

Many split-plot designs cannot be constructed directly from orthogonal arrays. However, you can create a split-plot design by using the `%MktEx` macro to create a design for the whole-plot factors and by using a second `%MktEx` step to append the subplot factors. The following steps show one way to create the whole-plot factors (applying techniques discussed previously) and add missing values or place holders for the additional subplot factors:

```
%mktorth(range=n=24, options=dups lineage)

data lev;
   set mktdeslev;
   where x2 = 14 and index(lineage, '2 ** 11 4 ** 1 6 ** 1');
   run;

%mktex(2 ** 14 6, n=24, out=whole(keep=x12-x14), options=nosort)
```

```
data in(keep=x1-x7);
   retain x1-x7 .;
   set whole;
   x1 = x13;
   x2 = x14;
   x3 = x12;
   run;

proc print; run;
```

These steps rely on creating two-level factors from a four-level factor that is orthogonal to a six-level factor. This is similar to logic used previously, but now a different parent design is used. The results are as follows:

```
Obs    x1    x2    x3    x4    x5    x6    x7

  1     1     1     1     .     .     .     .
  2     1     1     1     .     .     .     .
  3     1     1     1     .     .     .     .
  4     1     1     1     .     .     .     .
  5     1     1     1     .     .     .     .
  6     1     1     1     .     .     .     .

  7     1     2     2     .     .     .     .
  8     1     2     2     .     .     .     .
  9     1     2     2     .     .     .     .
 10     1     2     2     .     .     .     .
 11     1     2     2     .     .     .     .
 12     1     2     2     .     .     .     .

 13     2     2     1     .     .     .     .
 14     2     2     1     .     .     .     .
 15     2     2     1     .     .     .     .
 16     2     2     1     .     .     .     .
 17     2     2     1     .     .     .     .
 18     2     2     1     .     .     .     .

 19     2     1     2     .     .     .     .
 20     2     1     2     .     .     .     .
 21     2     1     2     .     .     .     .
 22     2     1     2     .     .     .     .
 23     2     1     2     .     .     .     .
 24     2     1     2     .     .     .     .
```

There are many other ways that you could make the initial data set with the whole plot factors, including more directly as follows:

```
data in(keep=x1-x7);
   retain x1-x7 .;
   input x1-x3;
   do i = 1 to 6; output; end;
   datalines;
1 1 1
1 2 2
2 2 1
2 1 2
;
```

This step reads an orthogonal array with no replication for the whole-plot factors and makes six copies of it. The following steps illustrate another way, this time by creating rather than reading an orthogonal array with no replication for the whole-plot factors:

```
%mktex(2 ** 3, n=4, options=nosort)

data in(keep=x1-x7);
   retain x1-x7 .;
   set design(rename=(x2=x1 x3=x2 x1=x3));
   do i = 1 to 6; output; end;
   run;
```

The `rename=` option is not necessary. In this example, it simply ensures that the levels match those used previously. This particular ordering of the first two whole-plot factors ensures minimal change of the factor levels across plots, which is desirable for some industrial uses of split-plot designs.

Regardless of how it is created, this design is used as an input initial design in a subsequent `%MktEx` run. In the `%MktEx` step, missing values are replaced, while nonmissing values are fixed and are not changed by the iterations. The whole-plot factors are input and never change, while the subplot factors are optimally (or at least nearly optimally) appended in the next step. The following steps append the subplot factors and evaluate and display the results:

```
%mktex(2 2 2   6 6 2 3, n=24, init=in, seed=104, options=nosort, maxiter=100)

%mkteval;

data splitplot;
   Plot = floor((_n_ - 1) / 6) + 1;
   set design;
   run;

proc print; id plot; by Plot;  run;
```

The results are as follows:

```
                 Canonical Correlations Between the Factors
             There is 1 Canonical Correlation Greater Than 0.316

               x1      x2      x3      x4      x5      x6      x7

         x1    1       0       0       0       0       0       0.10
         x2    0       1       0       0       0       0       0.10
         x3    0       0       1       0       0       0       0.19
         x4    0       0       0       1       0.50    0       0.29
         x5    0       0       0       0.50    1       0       0.29
         x6    0       0       0       0       0       1       0.10
         x7    0.10    0.10    0.19    0.29    0.29    0.10    1

              Canonical Correlations > 0.316 Between the Factors
             There is 1 Canonical Correlation Greater Than 0.316


                                      r    r Square

                      x4    x5    0.50      0.25

                         Summary of Frequencies
             There is 1 Canonical Correlation Greater Than 0.316
                       * - Indicates Unequal Frequencies

                         Frequencies

                 x1       12 12
                 x2       12 12
                 x3       12 12
                 x4       4 4 4 4 4 4
                 x5       4 4 4 4 4 4
                 x6       12 12
             *   x7       6 9 9
                 x1 x2    6 6 6 6
                 x1 x3    6 6 6 6
                 x1 x4    2 2 2 2 2 2 2 2 2 2 2 2
                 x1 x5    2 2 2 2 2 2 2 2 2 2 2 2
                 x1 x6    6 6 6 6
             *   x1 x7    3 5 4 3 4 5
         x2 x3    6 6 6 6
         x2 x4    2 2 2 2 2 2 2 2 2 2 2 2
         x2 x5    2 2 2 2 2 2 2 2 2 2 2 2
         x2 x6    6 6 6 6
   *     x2 x7    3 5 4 3 4 5
         x3 x4    2 2 2 2 2 2 2 2 2 2 2 2
         x3 x5    2 2 2 2 2 2 2 2 2 2 2 2
         x3 x6    6 6 6 6
   *     x3 x7    2 5 5 4 4 4
   *     x4 x5    0 1 0 1 1 1 1 0 1 1 0 1 1 1 0 0 1 1 0
                  1 1 0 1 1 1 1 1 1 0 0 1 0 1 1 1 0
         x4 x6    2 2 2 2 2 2 2 2 2 2 2 2
   *     x4 x7    1 2 1 1 1 2 1 2 1 1 2 1 1 1 2 1 1 2
         x5 x6    2 2 2 2 2 2 2 2 2 2 2 2
   *     x5 x7    1 1 2 1 2 1 1 1 2 1 1 2 1 2 1 1 2 1
   *     x6 x7    3 5 4 3 4 5
         N-Way    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                  1 1 1 1 1

      Plot    x1    x2    x3    x4    x5    x6    x7

         1     1     1     1     2     4     2     3
               1     1     1     5     2     2     2
               1     1     1     3     1     1     2
               1     1     1     4     3     1     3
               1     1     1     1     6     2     1
               1     1     1     6     5     1     2

         2     1     2     2     6     3     2     3
               1     2     2     5     4     1     3
               1     2     2     1     5     1     2
               1     2     2     3     2     2     1
               1     2     2     4     6     2     2
               1     2     2     2     1     1     1

         3     2     2     1     6     1     2     3
               2     2     1     5     3     1     1
               2     2     1     1     4     2     2
               2     2     1     3     5     2     3
               2     2     1     2     6     1     3
               2     2     1     4     2     1     2

         4     2     1     2     6     4     1     1
               2     1     2     3     6     1     2
               2     1     2     1     2     1     3
               2     1     2     2     3     2     2
               2     1     2     4     5     2     1
               2     1     2     5     1     2     3
```

The `%MktEval` macro automatically flags factors that are correlated with a canonical correlation greater than 0.316, which corresponds to $r^2 > 0.1$. Perfect orthogonality and balance are not possible since neither 6×6 nor 3×6 divides 24.

Split-Plot Designs with Better Within-Plot Balance

If balance in the subplot factors within the plots is a concern, then you can often force better balance by adding the plot number as a factor for the duration of the design creation, for example as follows:

```
data in(keep=x1-x8);
   retain x1-x8 .;
   set whole; /* same design with whole-plot factors as before */
   x1 = x13;
   x2 = x14;
   x3 = x12;
   x4 = floor((_n_ - 1) / 6) + 1;  /* Plot number: 1, 2, 3, 4 */
   run;

%mktex(2 2 2   4   6 6 2 3, n=24, init=in, seed=104, options=nosort,
       maxiter=100, ridge=1e-4)

%mkteval;

proc print label; by x4; id x4; label x4 = 'Plot'; run;
```

This results in a design with a linear dependency and hence zero efficiency[*], however, that does not pose a problem for the `%MktEx` macro. It iterates and finds a good design optimizing a ridged efficiency criterion (a constant value is added to the diagonal of the $\mathbf{X}'\mathbf{X}$ matrix to make it nonsingular). The final design, when evaluated without the plot factor, has a reasonable *D*-efficiency. The `ridge=` option is not necessary in this example. Without it, the default is `ridge=1e-7`. However, if the addition of the plot factor results in more parameters than runs, then you must specify the `ridge=` option. The macro will not produce a design with more parameters than runs without an explicit `ridge=` specification. The results are as follows:

```
                 Canonical Correlations Between the Factors
             There are 4 Canonical Correlations Greater Than 0.316

             x1      x2      x3      x4      x5      x6      x7      x8

      x1     1       0       0       1.00    0       0       0       0
      x2     0       1       0       1.00    0       0       0       0
      x3     0       0       1       1.00    0       0       0       0
      x4     1.00    1.00    1.00    1       0       0       0       0
      x5     0       0       0       0       1       0.43    0       0.25
      x6     0       0       0       0       0.43    1       0       0.25
      x7     0       0       0       0       0       0       1       0
      x8     0       0       0       0       0.25    0.25    0       1
```

[*]In this design, the plot variable equals $2\times$`x1` + `x3` $- 2$.

```
                Canonical Correlations > 0.316 Between the Factors
              There are 4 Canonical Correlations Greater Than 0.316


                                           r    r Square

                           x1    x4    1.00      1.00
                           x2    x4    1.00      1.00
                           x3    x4    1.00      1.00
                           x5    x6    0.43      0.19

                              Summary of Frequencies
              There are 4 Canonical Correlations Greater Than 0.316
                         * - Indicates Unequal Frequencies

                             Frequencies

                  x1       12 12
                  x2       12 12
                  x3       12 12
                  x4       6 6 6 6
                  x5       4 4 4 4 4 4
                  x6       4 4 4 4 4 4
                  x7       12 12
                  x8       8 8 8
                  x1 x2    6 6 6 6
                  x1 x3    6 6 6 6
               *  x1 x4    6 6 0 0 0 0 6 6
                  x1 x5    2 2 2 2 2 2 2 2 2 2 2 2
                  x1 x6    2 2 2 2 2 2 2 2 2 2 2 2
                  x1 x7    6 6 6 6
                  x1 x8    4 4 4 4 4 4
                  x2 x3    6 6 6 6
               *  x2 x4    6 0 0 6 0 6 6 0
                  x2 x5    2 2 2 2 2 2 2 2 2 2 2 2
                  x2 x6    2 2 2 2 2 2 2 2 2 2 2 2
                  x2 x7    6 6 6 6
                  x2 x8    4 4 4 4 4 4
               *  x3 x4    6 0 6 0 0 6 0 6
                  x3 x5    2 2 2 2 2 2 2 2 2 2 2 2
                  x3 x6    2 2 2 2 2 2 2 2 2 2 2 2
                  x3 x7    6 6 6 6
                  x3 x8    4 4 4 4 4 4
                  x4 x5    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                           1 1 1 1 1
                  x4 x6    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                           1 1 1 1 1
                  x4 x7    3 3 3 3 3 3 3 3
                  x4 x8    2 2 2 2 2 2 2 2 2 2 2 2
```

```
*    x5 x6    1 1 0 1 0 1 1 0 0 1 1 1 0 1 1 1 1 0 1
              1 1 0 1 0 1 1 1 0 0 1 0 0 1 1 1 1
     x5 x7    2 2 2 2 2 2 2 2 2 2 2 2
*    x5 x8    2 1 1 1 1 2 1 2 1 1 1 2 1 2 1 2 1 1
     x6 x7    2 2 2 2 2 2 2 2 2 2 2 2
*    x6 x8    1 1 2 1 2 1 2 1 1 2 1 1 1 1 2 1 2 1
     x7 x8    4 4 4 4 4 4
     N-Way    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
              1 1 1 1 1

    Plot   x1    x2    x3    x5    x6    x7    x8

      1     1     1     1     4     5     1     3
            1     1     1     3     4     1     2
            1     1     1     2     1     2     3
            1     1     1     1     2     2     1
            1     1     1     6     3     1     1
            1     1     1     5     6     2     2

      2     1     2     2     2     6     1     3
            1     2     2     4     3     2     3
            1     2     2     6     4     1     1
            1     2     2     1     1     1     2
            1     2     2     3     5     2     1
            1     2     2     5     2     2     2

      3     2     2     1     4     1     2     1
            2     2     1     3     2     1     3
            2     2     1     5     3     1     1
            2     2     1     6     6     2     2
            2     2     1     2     5     1     2
            2     2     1     1     4     2     3

      4     2     1     2     4     2     1     2
            2     1     2     5     1     1     3
            2     1     2     6     5     2     3
            2     1     2     1     6     1     1
            2     1     2     3     3     2     2
            2     1     2     2     4     2     1
```

Now, all of the factors are balanced within each plot. This can be seen by examining the two-way frequencies involving `x4` along with `x5` through `x8`. This design in fact looks better than the one found previously, although in general, there is no guarantee that this approach will make a better design. Ignoring `x4`, the plot number, all of the nonzero canonical correlations have gotten smaller in this design compared to what they were previously. You can evaluate the design ignoring the plot number by using this step: `%mkteval(data=design(drop=x4))`.

Candidate Set Search

The candidate-set search has two parts. First, either PROC PLAN is run to create a full-factorial design for small problems, or PROC FACTEX is run to create a fractional-factorial design for large problems. Either way, this design is a candidate set that in the second part is searched by PROC OPTEX using the modified Fedorov algorithm. A design is built from a selection of the rows of the candidate set (Fedorov 1972; Cook and Nachtsheim 1980). The modified Fedorov algorithm considers each run in the design and each candidate run. Candidate runs are swapped in and design runs are swapped out if the swap improves *D*-efficiency.

Coordinate Exchange

The `%MktEx` macro also uses the coordinate-exchange algorithm, based on Meyer and Nachtsheim (1995). The coordinate-exchange algorithm considers each level of each factor, and considers the effect on *D*-efficiency of changing a level ($1 \rightarrow 2$, or $1 \rightarrow 3$, or $2 \rightarrow 1$, or $2 \rightarrow 3$, or $3 \rightarrow 1$, or $3 \rightarrow 2$, and so on). Exchanges that increase efficiency are performed. Typically, the macro first tries to initialize the design with an orthogonal design (`Tab` refers to the orthogonal array table or catalog) and a random design (`Ran`) both. Levels that are not orthogonally initialized can be exchanged for other levels if the exchange increases efficiency.

The initialization might be more complicated. Say you asked for the design $4^1 5^1 3^5$ in 18 runs. The macro would use the orthogonal design $3^6 6^1$ in 18 runs to initialize the three-level factors orthogonally, and the five-level factor with the six-level factor coded down to five levels (which is imbalanced). The four-level factor would be randomly initialized. The macro would also try the same initialization but with a random rather than unbalanced initialization of the five-level factor, as a minor variation on the first initialization. In the next initialization variation, the macro would use a fully random initialization. If the number of runs requested were smaller than the number or runs in the initial orthogonal design, the macro would initialize the design with just the first n rows of the orthogonal design. Similarly, if the number of runs requested were larger than the number or runs in the initial orthogonal design, the macro would initialize part of the design with the orthogonal design and the remaining rows and columns randomly. The coordinate-exchange algorithm considers each level of each factor that is not orthogonally initialized, and it exchanges a level if the exchange improves *D*-efficiency. When the number or runs in the orthogonal design does not match the number of runs desired, none of the design is initialized orthogonally.

The coordinate-exchange algorithm is not restricted by having a candidate set and hence can *potentially* consider every possible design. That is, no design is precluded from consideration due to the limitations of a candidate set. In practice, however, both the candidate-set-based and coordinate-exchange algorithms consider only a *tiny* fraction of the possible designs. When the number of runs in the full-factorial design is very small (say 100 or 200 runs), the modified Fedorov algorithm and coordinate exchange algorithms usually work equally well. When the number of runs in the full-factorial design is small (up to several thousand), the modified Fedorov algorithm is usually superior to coordinate exchange, particularly in finding designs with interactions. When the full-factorial design is larger, coordinate exchange is usually the superior approach. However, heuristics like these are often wrong, which is why the macro tries both methods to see which one is really best for each problem.

Next, the `%MktEx` macro determines which algorithm (candidate set search, coordinate exchange with partial orthogonal initialization, or coordinate exchange with random initialization) is working best and tries more iterations using that approach. It starts by displaying the initial (`Ini`) best efficiency.

Next, the `%MktEx` macro tries to improve the best design it found previously. Using the previous best design as an initialization (`Pre`), and random mutations of the initialization (`Mut`) and simulated annealing (`Ann`), the macro uses the coordinate-exchange algorithm to try to find a better design. This step is important because the best design that the macro found might be an intermediate design and might not be the final design at the end of an iteration. Sometimes, the iterations deliberately make the designs less efficient, and sometimes, the macro never finds a design as efficient or more efficient again. Hence, it is worthwhile to see if the best design found so far can be improved. At the end, PROC OPTEX is called to display the levels of each factor and the final *D*-efficiency.

Random mutations involve adding random noise to the initial design before iterations start (levels are randomly changed). This might eliminate the perfect balance that will often be in the initial design. By default, random mutations are used with designs with fully random initializations and in the design refinement step; orthogonal initial designs are not mutated.

Coordinate exchange can be combined with the simulated annealing optimization technique (Kirkpatrick, Gellat, and Vecchi 1983). Annealing refers to the cooling of a liquid in a heat bath. The structure of the solid depends on the rate of cooling. Coordinate exchange without simulated annealing seeks to maximize *D*-efficiency at every step. Coordinate exchange with simulated annealing lets *D*-efficiency occasionally decrease with a probability that decreases with each iteration. This is analogous to slower cooling, and it helps overcome local optima.

For design 1, for the first level of the first factor, by default, the macro might execute an exchange (say change a 2 to a 1) that makes the design worse with probability 0.05. As more and more exchanges occur, this probability decreases so at the end of the processing of design 1, exchanges that decrease efficiency are hardly ever done. For design 2, this same process is repeated, again starting by default with an annealing probability of 0.05. This often helps the algorithm overcome local efficiency maxima. To envision this, imagine that you are standing on a molehill next to a mountain. The only way you can start going up the mountain is to first step down off the molehill. Once you are on the mountain, you might occasionally hit a dead end, where all you can do is step down and look for a better place to continue going up. Simulated annealing, by occasionally stepping down the efficiency function, often lets the macro go farther up it than it would otherwise. The simulated annealing is why you will sometimes see designs getting worse in the iteration history. The macro keeps track of the best design, not the final design in each step. By default, annealing is used with designs with fully random initializations and in the design refinement step. Simulated annealing is not used with orthogonally initialized designs.

Aliasing Structure

The following example illustrates the `examine=aliasing=2` option:

```
%mktex(3 ** 4, n=9, examine=aliasing=2)
```

The preceding step produces the following results:

```
                                    Aliasing Scheme
Estimable  Aliased
Effect     Effects

Intercept

x1         x2*x3 x2*x4 x3*x4

x2         x1*x3 x1*x4 x3*x4

x3         x1*x2 x1*x4 x2*x4

x4         x1*x2 x1*x3 x2*x3

NOTE: Some parameters in the estimable effects are aliased with some parameters
 in the aliased effects.  For effects with more than two levels, the aliasing
 scheme displayed here is potentially partial.  Specify EXAMINE=FULL ALIASING=n
 to see the full aliasing structure.
```

These results show that one or both of the two parameters in each of `x1-x4` are aliased with one or more of the $(3 - 1) \times (3 - 1) = 4$ parameters in the each of three two-way interactions that do not involve the estimable effect.

You can get the full results as follows:

```
%mktex(3 ** 4, n=9, examine=aliasing=2 full)
```

The full aliasing results involving the `x1` main effect are as follows:

```
x11 - x21x32 - x22x31 + 0.5*x21x41 - 0.5*x21x42 + 0.5*x22x41 + 1.5*x22x42 +
   0.5*x31x41 - 0.5*x31x42 - 0.5*x32x41 - 1.5*x32x42

x12 - 0.3333*x21x31 + x22x32 - 0.1667*x21x41 - 0.5*x21x42 + 0.5*x22x41 -
   0.5*x22x42 + 0.1667*x31x41 + 0.5*x31x42 + 0.5*x32x41 - 0.5*x32x42
```

These results show that the level 1 parameter of `x1` cannot be estimated independently of the interaction term involving `x2` level 1 and `x3` level 2, the interaction term involving `x2` level 2 and `x3` level 1, the interaction term involving `x2` level 1 and `x4` level 1, the interaction term involving `x2` level 1 and `x4` level 2, the interaction term involving `x2` level 2 and `x4` level 1, and so on.

Note, however, that even these results are not really the full results because only two-way interactions are included. To really see the full extent of the aliasing, you need to add all three-way and the four-way interaction as follows:

```
%mktex(3 ** 4, n=9, examine=aliasing=4 full)
```

The first part of the full aliasing structure is as follows:

```
                                Aliasing Structure

Intercept - 0.6667*x11x21x32 - 0.6667*x11x22x31 - 0.6667*x12x21x31 +
   2*x12x22x32 + 0.3333*x11x21x41 - 0.3333*x11x21x42 + 0.3333*x11x22x41 +
   x11x22x42 - 0.3333*x12x21x41 - x12x21x42 + x12x22x41 - x12x22x42 +
   0.3333*x11x31x41 - 0.3333*x11x31x42 - 0.3333*x11x32x41 - x11x32x42 +
   0.3333*x12x31x41 + x12x31x42 + x12x32x41 - x12x32x42 + 0.3333*x21x31x41 -
   0.3333*x21x31x42 + 0.3333*x21x32x41 + x21x32x42 - 0.3333*x22x31x41 -
   x22x31x42 + x22x32x41 - x22x32x42

x11 - x21x32 - x22x31 + 0.3333*x11x21x31 - x11x22x32 + x12x21x32 + x12x22x31 +
   0.5*x21x41 - 0.5*x21x42 + 0.5*x22x41 + 1.5*x22x42 + 0.1667*x11x21x41 +
   0.5*x11x21x42 - 0.5*x11x22x41 + 0.5*x11x22x42 - 0.5*x12x21x41 +
   0.5*x12x21x42 - 0.5*x12x22x41 - 1.5*x12x22x42 + 0.5*x31x41 - 0.5*x31x42 -
   0.5*x32x41 - 1.5*x32x42 - 0.1667*x11x31x41 - 0.5*x11x31x42 - 0.5*x11x32x41 +
   0.5*x11x32x42 - 0.5*x12x31x41 + 0.5*x12x31x42 + 0.5*x12x32x41 +
   1.5*x12x32x42 + 0.3333*x11x21x31x41 - 0.3333*x11x21x31x42 +
   0.3333*x11x21x32x41 + x11x21x32x42 - 0.3333*x11x22x31x41 - x11x22x31x42 +
   x11x22x32x41 - x11x22x32x42

x12 - 0.3333*x21x31 + x22x32 + 0.3333*x11x21x32 + 0.3333*x11x22x31 -
   0.3333*x12x21x31 + x12x22x32 - 0.1667*x21x41 - 0.5*x21x42 + 0.5*x22x41 -
   0.5*x22x42 - 0.1667*x11x21x41 + 0.1667*x11x21x42 - 0.1667*x11x22x41 -
   0.5*x11x22x42 - 0.1667*x12x21x41 - 0.5*x12x21x42 + 0.5*x12x22x41 -
   0.5*x12x22x42 + 0.1667*x31x41 + 0.5*x31x42 + 0.5*x32x41 - 0.5*x32x42 -
   0.1667*x11x31x41 + 0.1667*x11x31x42 + 0.1667*x11x32x41 + 0.5*x11x32x42 +
   0.1667*x12x31x41 + 0.5*x12x31x42 + 0.5*x12x32x41 - 0.5*x12x32x42 +
   0.3333*x12x21x31x41 - 0.3333*x12x21x31x42 + 0.3333*x12x21x32x41 +
   x12x21x32x42 - 0.3333*x12x22x31x41 - x12x22x31x42 + x12x22x32x41 -
   x12x22x32x42
```

The aliasing scheme, which is the new summary, is as follows:

```
                                  Aliasing Scheme
Estimable  Aliased
Effect     Effects

Intercept  x1*x2*x3 x1*x2*x4 x1*x3*x4 x2*x3*x4

x1         x2*x3 x2*x4 x3*x4 x1*x2*x3 x1*x2*x4 x1*x3*x4 x1*x2*x3*x4

x2         x1*x3 x1*x4 x3*x4 x1*x2*x3 x1*x2*x4 x2*x3*x4 x1*x2*x3*x4

x3         x1*x2 x1*x4 x2*x4 x1*x2*x3 x1*x3*x4 x2*x3*x4 x1*x2*x3*x4

x4         x1*x2 x1*x3 x2*x3 x1*x2*x4 x1*x3*x4 x2*x3*x4 x1*x2*x3*x4

NOTE: Some parameters in the estimable effects are aliased with some parameters
 in the aliased effects.  For effects with more than two levels, the aliasing
 scheme displayed here is potentially partial.  Specify EXAMINE=FULL ALIASING=n
 to see the full aliasing structure.
```

For the complicated designs that we use in practice, particularly for choice models, the aliasing structure and aliasing scheme are often very long and complicated. In many cases, they cannot be displayed because the size of the model gets unwieldy with m main effect parameters, $m(m-1)/2$ two-way interaction parameters, and so on.

We can use the `%MktEx` macro to create a resolution IV design (all main effects are estimable free of each other and free of all two-factor interactions, but some two-factor interactions are confounded with other two-factor interactions), for m two-level factors (where m is a multiple of 4), and evaluate it as follows:

```
%let m = 12;
%mktorth(range=n=2 * &m, options=lineage dups, maxlev=&m)

%mktex(2 ** &m, n=2 * &m, examine=aliasing=2,
       cat=mktdeslev(where=(index(compbl(design), "2 ** &m &m ** 1"))))
```

This example uses the `%MktOrth` macro to create the instructions for creating all designs in $2m$ runs, and then sends the `%MktEx` macro instructions for just the design with m two-level factors and an m-level factor, which produces the desired design. Often, the `%MktEx` macro has many ways to make a specified design and will not choose the one you have in mind unless you give it specific instructions such as we did here. This step creates a design from the first m columns of a design of the form:

$$\begin{array}{rc} \mathbf{H}_m & \ell_m \\ -\mathbf{H}_m & \ell_m \end{array}$$

$\mathbf{H}_m$ is a Hadamard matrix of order m, and ℓ_m is a column vector where $\ell_m' = [0\ 1\ 2\ ...\ m-1]$. The resulting design has m two-level factors in $2m$ runs and has the following aliasing scheme:

```
                               Aliasing Scheme
Estimable  Aliased
Effect     Effects

Intercept

x1

x2

x3

x4

x5

x6

x7

x8

x9

x10

x11

x12

x1*x2      x1*x7 x1*x8 x1*x11 x1*x12 x2*x7 x2*x8 x2*x11 x2*x12 x3*x7 x3*x8
           x3*x9 x3*x11 x3*x12 x4*x7 x4*x8 x4*x10 x4*x11 x4*x12 x5*x6 x5*x7
           x5*x8 x5*x11 x5*x12 x6*x7 x6*x8 x6*x11 x6*x12 x7*x9 x7*x10 x8*x9
           x8*x10 x9*x11 x9*x12 x10*x11 x10*x12

x1*x3      x1*x7 x1*x8 x1*x10 x1*x11 x2*x6 x2*x7 x2*x8 x2*x10 x2*x11 x3*x7
           x3*x8 x3*x10 x3*x11 x4*x7 x4*x8 x4*x10 x4*x11 x4*x12 x5*x7 x5*x8
           x5*x9 x5*x10 x5*x11 x6*x7 x6*x8 x6*x10 x6*x11 x7*x9 x7*x12 x8*x9
           x8*x12 x9*x10 x9*x11 x10*x12 x11*x12

x1*x4      x1*x7 x1*x9 x1*x10 x1*x11 x2*x7 x2*x9 x2*x10 x2*x11 x2*x12 x3*x6
           x3*x7 x3*x9 x3*x10 x3*x11 x4*x7 x4*x9 x4*x10 x4*x11 x5*x7 x5*x8
           x5*x9 x5*x10 x5*x11 x6*x7 x6*x9 x6*x10 x6*x11 x7*x8 x7*x12 x8*x9
           x8*x10 x8*x11 x9*x12 x10*x12 x11*x12

x1*x5      x1*x7 x1*x9 x1*x11 x1*x12 x2*x7 x2*x9 x2*x10 x2*x11 x2*x12 x3*x7
           x3*x8 x3*x9 x3*x11 x3*x12 x4*x6 x4*x7 x4*x9 x4*x11 x4*x12 x5*x7
           x5*x9 x5*x11 x5*x12 x6*x7 x6*x9 x6*x11 x6*x12 x7*x8 x7*x10 x8*x9
           x8*x11 x8*x12 x9*x10 x10*x11 x10*x12
```

```
    x1*x6      x1*x8 x1*x9 x1*x10 x1*x11 x1*x12 x2*x6 x2*x8 x2*x9 x2*x10 x2*x11
               x2*x12 x3*x6 x3*x8 x3*x9 x3*x10 x3*x11 x3*x12 x4*x6 x4*x8 x4*x9
               x4*x10 x4*x11 x4*x12 x5*x6 x5*x8 x5*x9 x5*x10 x5*x11 x5*x12 x6*x7
               x7*x8 x7*x9 x7*x10 x7*x11 x7*x12

    x1*x7      x1*x8 x1*x9 x1*x10 x1*x11 x1*x12 x2*x3 x2*x4 x2*x5 x2*x6 x2*x7 x2*x8
               x2*x9 x2*x10 x2*x11 x2*x12 x3*x4 x3*x5 x3*x6 x3*x7 x3*x8 x3*x9
               x3*x10 x3*x11 x3*x12 x4*x5 x4*x6 x4*x7 x4*x8 x4*x9 x4*x10 x4*x11
               x4*x12 x5*x6 x5*x7 x5*x8 x5*x9 x5*x10 x5*x11 x5*x12 x6*x7 x6*x8
               x6*x9 x6*x10 x6*x11 x6*x12 x7*x8 x7*x9 x7*x10 x7*x11 x7*x12 x8*x9
               x8*x10 x8*x11 x8*x12 x9*x10 x9*x11 x9*x12 x10*x11 x10*x12 x11*x12

    NOTE: Some parameters in the estimable effects are aliased with some parameters
     in the aliased effects.  For effects with more than two levels, the aliasing
     scheme displayed here is potentially partial.  Specify EXAMINE=FULL ALIASING=n
     to see the full aliasing structure.
```

The aliasing scheme shows that we did in fact find a resolution IV design with all main effects estimable free of each other and free of all two-factor interactions. You can see by looking at the interaction terms in the estimable effects and those that are part of the `x1*x7` aliasing scheme that in this case at least part of every two-way interaction is aliased with at least part of some other two-way interactions.

%MktEx Macro Notes

The `%MktEx` macro displays notes in the SAS log to show you what it is doing while it is running. Most of the notes that would normally come out of the macro's procedure and DATA steps are suppressed by default by an `options nonotes` statement. This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro. This section describes the notes that are normally not suppressed.

The macro will usually start by displaying one of the following notes (filling in a value after `n=`):

```
   NOTE: Generating the Hadamard design, n=.
   NOTE: Generating the full-factorial design, n=.
   NOTE: Generating the fractional-factorial design, n=.
   NOTE: Generating the orthogonal array design, n=.
```

These messages tell you which type of orthogonal design the macro is constructing. The design might be the final design, or it might provide an initialization for the coordinate exchange algorithm. In some cases, it might not have the same number of runs, `n`, as the final design. Usually this step is fast, but constructing some fractional-factorial designs might be time consuming.

If the macro is going to use PROC OPTEX to search a candidate set, it will display the following note:

```
   NOTE: Generating the candidate set.
```

This step will usually be fast. Next, when a candidate set is searched, the macro will display the following note, substituting in values for the ellipses:

```
   NOTE: Performing ... searches of ... candidates.
```

This step might take a while depending on the size of the candidate set and the size of the design. When there are a lot of restrictions and a fractional-factorial candidate set is being used, the candidate set might be so restricted that it does not contain enough information to make the design. In that case, you will get the following message:

```
NOTE: The candidate-set initialization failed,
      but the MKTEX macro is continuing.
```

Even though part of the macro's algorithm failed, it is *not* a problem. The macro just goes on to the coordinate-exchange algorithm, which will almost certainly work better than searching any severely-restricted candidate set.

For large designs, you usually will want to skip the PROC OPTEX iterations. The macro might display the following note:

```
NOTE: With a design this large, you may get faster results with OPTITER=0.
```

Sometimes you will get the following note:

```
NOTE: Stopping since it appears that no improvement is possible.
```

When the macro keeps finding the same maximum *D*-efficiency over and over again in different designs, it might stop early. This might mean that the macro has found the optimal design, or it might mean that the macro keeps finding a very attractive local optimum. Either way, it is unlikely that the macro will do any better. You can control this using the `stopearly=` option.

The macro has options that control the amount of time it spends trying different techniques. When time expires, the macro might switch to other techniques before it completes the usual maximum number of iterations. When this happens, the macro tells you with the following notes:

```
NOTE: Switching to a random initialization after ... minutes and
      ... designs.
NOTE: Quitting the algorithm search after ... minutes and ... designs.
NOTE: Quitting the design search after ... minutes and ... designs.
NOTE: Quitting the refinement step after ... minutes and ... designs.
```

When there are restrictions, or when you specify that you do not want duplicate runs, you can also specify `options=accept`. This means that you are willing to accept designs that violate the restrictions. With `options=accept`, the macro will tell you if the restrictions are not met with the following notes:

```
NOTE: The restrictions were not met.
NOTE: The design has duplicate runs.
```

`%MktEx` optimizes a ridged efficiency criterion, that is, a small number is added to the diagonal of $(\mathbf{X}'\mathbf{X})^{-1}$. Usually, the ridged criterion is virtually the same as the unridged criterion. When `%MktEx` detects that this is not true, it displays the following notes:

```
NOTE: The final   ridged D-efficiency criterion is ....
NOTE: The final unridged D-efficiency criterion is ....
```

The macro ends with one of the following two messages:

```
NOTE: The MKTEX macro used ... seconds.
NOTE: The MKTEX macro used ... minutes.
```

%MktEx Macro Iteration History

This section provides information about interpreting the iteration history table produced by the `%MktEx` macro. Some of the results are as follows:

```
                       Algorithm Search History

                              Current           Best
Design     Row,Col  D-Efficiency  D-Efficiency  Notes
----------------------------------------------------------
     1      Start       82.2172       82.2172  Can
     1        End       82.2172

     2      Start       78.5039                Tab,Ran
     2      5  14       83.2098       83.2098
     2      6  14       83.3917       83.3917
     2      6  15       83.5655       83.5655
     2      7  14       83.7278       83.7278
     2      7  15       84.0318       84.0318
     2      7  15       84.3370       84.3370
     2      8  14       85.1449       85.1449
     .
     .
     .
     2        End       98.0624
     .
     .
     .
    12      Start       51.8915                Ran,Mut,Ann
    12        End       93.0214
     .
     .
     .


                        Design Search History

                              Current           Best
Design     Row,Col  D-Efficiency  D-Efficiency  Notes
----------------------------------------------------------
     0    Initial       98.8933       98.8933  Ini

     1      Start       80.4296                Tab,Ran
     1        End       98.8567
     .
     .
     .
```

```
                      Design Refinement History

                                 Current          Best
          Design    Row,Col  D-Efficiency  D-Efficiency  Notes
          ----------------------------------------------------------
               0    Initial       98.9438       98.9438  Ini

               1      Start       94.7490                Pre,Mut,Ann
               1        End       92.1336
               .
               .
               .
```

The first column, `Design`, is a design number. Each design corresponds to a complete iteration using a different initialization. Initial designs are numbered zero. The second column is `Row,Col`, which shows the design row and column that is changing in the coordinate-exchange algorithm. This column also contains `Start` for displaying the initial efficiency, `End` for displaying the final efficiency, and `Initial` for displaying the efficiency of a previously created initial design (perhaps created externally or perhaps created in a previous step). The `Current D-Efficiency` column contains the *D*-efficiency for the design including starting, intermediate and final values. The next column is `Best D-Efficiency`. Values are put in this column for initial designs and when a design is found that is as good as or better than the previous best design. The last column, `Notes`, contains assorted algorithm and explanatory details. Values are added to the table at the beginning of an iteration, at the end of an iteration, when a better design is found, and when a design first conforms to restrictions. The note `Conforms` is displayed when a design first conforms to the restrictions. From then on, the design continues to conform even though `Conforms` is not displayed in every line. Details of the candidate search iterations are not shown. Only the *D*-efficiency for the best design found through candidate search is shown.

The notes are as follows:

`Can`	– the results of a candidate-set search
`Tab`	– design table or catalog (orthogonal array, full, or fractional-factorial) initialization (full or in part)
`Ran`	– random initialization (full or in part)
`Unb`	– unbalanced initial design (usually in part)
`Ini`	– initial design
`Mut`	– random mutations of the initial design were performed
`Ann`	– simulated annealing was used in this iteration
`Pre`	– using previous best design as a starting point
`Conforms`	– design conforms to restrictions
`Violations`	– number of restriction violations

Often, more than one note appears. For example, the triples `Ran,Mut,Ann` and `Pre,Mut,Ann` frequently appear together.

The iteration history consists of three tables.

`Algorithm Search History`	– searches for a design and the best algorithm for this problem
`Design Search History`	– read the order from a data set
`Design Refinement History`	– tries to refine the best design

%MktEx Macro Options

The following options can be used with the `%MktEx` macro:

Option	Description
`list`	(positional) list of the numbers of factor levels
	(positional) “help” or “?” displays syntax summary
`anneal=`*n1* < *n2* < *n3* >>	starting probability for annealing
`annealfun=`*function*	annealing probability function
`anniter=`*n1* < *n2* < *n3* >>	first annealing iteration
`balance=`*n*	maximum level-frequency range
`big=`*n* < *choose* >	size of big full-factorial design
`canditer=`*n1* < *n2* >	iterations for OPTEX designs
`cat=`*SAS-data-set*	input design catalog
`detfuzz=`*n*	determinants change increment
`examine=`< I > < V > <*aliasing* >	matrices that you want to examine
`exchange=`*n*	number of factors to exchange
`fixed=`*variable*	indicates runs that are fixed
`holdouts=`*n*	adds holdout observations
`imlopts=`*options*	IML PROC statement options
`init=`*SAS-data-set*	initial input experimental design
`interact=`*interaction-list*	interaction terms
`iter=`*n1* < *n2* < *n3* >>	maximum number of iterations
`levels=`*value*	assigning final factor levels
`maxdesigns=`*n*	maximum number of designs to make
`maxiter=`*n1* < *n2* < *n3* >>	maximum number of iterations
`maxstages=`*n*	maximum number of algorithm stages
`maxtime=`*n1* < *n2* < *n3* >>	approximate maximum run time
`mintry=`*n*	minimum number of rows to process
`mutate=`*n1* < *n2* < *n3* >>	mutation probability
`mutiter=`*n1* < *n2* < *n3* >>	first iteration to consider mutating
`n=`*n*	number of runs in the design
`options=accept`	accept designs that violate restrictions
`options=check`	checks the efficiency of the `init=` design
`options=file`	renders the design to a file
`options=int`	add an intercept variable `x0` to the design
`options=justinit`	stop processing after making the initial design
`options=largedesign`	stop after `maxtime=` minutes have elapsed
`options=lineage`	displays the lineage of the orthogonal array
`options=nodups`	eliminates duplicate runs
`options=nofinal`	skips displaying the final efficiency
`options=nohistory`	does not display the iteration history
`options=nooadups`	check for orthogonal array with duplicate runs
`options=noqc`	do not use the SAS/QC product
`options=nosort`	does not sort the design
`options=nox`	suppress the the `x`*n* scalars with restrictions
`options=quick`	`optiter=0`, `maxdesigns=2`, `unbalanced=0`, `tabiter=1`
`options=quickr`	`optiter=0`, `maxdesigns=1`, `unbalanced=0`, `tabiter=0`
`options=quickt`	`optiter=0`, `maxdesigns=1`, `unbalanced=0`, `tabiter=1`

Option	Description
`options=render`	displays the design compactly in the SAS listing
`options=refine`	with `init=`, never reinitializes
`options=resrep`	reports on the progress of the restrictions
`options=+-`	renders –1 as – and 1 as + in two-level factors
`options=3`	applies `options=+-` to 3-level factors
`options=512`	adds some designs in 512 runs
`optiter=`*n1* < *n2* >	OPTEX iterations
`order=`*value*	coordinate exchange column order
`out=`*SAS-data-set*	output experimental design
`outall=`*SAS-data-set*	output data set with all designs
`outeff=`*SAS-data-set*	output data set with final efficiency
`outr=`*SAS-data-set*	randomized output experimental design
`partial=`*n*	partial-profile design
`repeat=`*n1 n2 n3*	times to iterate on a row
`reslist=`*list*	constant matrix list
`resmac=`*macro-name*	constant matrix creation macro
`restrictions=`*macro-name*	restrictions macro
`ridge=`*n*	ridging factor
`seed=`*n*	random number seed
`stopearly=`*n*	the macro can stop early
`tabiter=`*n1* < *n2* >	design table initialization iterations
`tabsize=`*n*	orthogonal array size
`target=`*n*	target efficiency criterion
`unbalanced=`*n1* < *n2* >	unbalance initial design iterations

Help Option

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%mktex(help)
%mktex(?)
```

Required Options

The `n=` options is required, and the `list` option is almost always required.

list

specifies a list of the numbers of levels of all the factors. For example, for 3 two-level factors specify either `2 2 2` or `2 ** 3`. Lists of numbers, like `2 2 3 3 4 4` or a *levels**number of factors* syntax like: `2**2 3**2 4**2` can be used, or both can be combined: `2 2 3**4 5 6`. The specification `3**4` means 4 three-level factors. Note that the factor list is a positional parameter. This means that if it is specified, it must come first, and unlike all other parameters, it is not specified after a name and an equal sign. Usually, you have to specify a list. However, in some cases, you can just specify `n=` and omit the list and a default list is implied. For example, `n=18` implies a list of `2 3 ** 7`. When the list is omitted, and if there are no interactions, restrictions, or duplicate exclusions, then by default there are no OPTEX iterations (`optiter=0`).

n= *n*

specifies the number of runs in the design. You must specify `n=`. The following example uses the `%MktRuns` macro to get suggestions for values of `n=`:

```
%mktruns(4 2 ** 5 3 ** 5)
```

In this case, this macro suggests several sizes including orthogonal designs with `n=72` and `n=144` runs and some smaller nonorthogonal designs including `n=36, 24, 48, 60`.

Basic Options

This next group of options contains some of the more commonly used options.

balance= *n*

specifies the maximum allowable level-frequency range. You use this option to tell the macro that it should make an extra effort to ensure that the design is balanced or at least nearly balanced. Specify a positive integer, usually 1 or 2, that specifies the degree of imbalance that is acceptable. The `balance=n` option specifies that for each factor, a difference between the frequencies of the most and least frequently occurring levels should be no larger than n.

When you specify `balance=`, particularly if you specify `balance=0`, you should also specify `mintry=` (perhaps something like `mintry=5 * n`, or `mintry=10 * n`). When `balance=` and `mintry=mt` are both specified, then the balance restrictions are ignored for the first `mt - 3 * n / 2` passes through the design. During this period, the badness function for the balance restrictions is set to 1 so that `%MktEx` knows that the design does not conform. After that, all restrictions are considered. The `balance=` option works best when its restrictions are imposed on a reasonably efficient design not an inefficient initial design. You can specify `balance=0`, without specify `mintry=`, however, this might not be a good idea because the macro needs the flexibility to have imbalance as it refines the design. Often, the design actually found will be better balanced than your `balance=n` specification would require. For this reason, it is good to start by specifying a value larger than the minimum acceptable value. The larger the value, the more freedom the algorithm has to optimize both balance and efficiency.

The `balance=` option works by adding restrictions to the design. The badness of each column (how far each column is from conforming to the balance restrictions) is evaluated and the results stored in a scalar `__bbad`. When you specify other restrictions, this is added to the `bad` value created by your restrictions macro. You can use your restrictions macro to change or differentially weight `__bbad` before the final addition of the components of design badness takes place (see page 1065).

The `%MktEx` macro usually does a good job of producing nearly balanced designs, but if balance is critically important, and your designs are not balanced enough, you can sometimes achieve better balance by specifying `balance=`, but usually at the price of worse efficiency, sometimes much worse. By default, no additional restrictions are added. Another approach is to instead use the `%MktBal` macro, which for main effects plans with no restrictions, produces designs that are guaranteed to have optimal balance.

examine= < `I` > < `V` > < `aliasing=`*n* > < `full` > < `main` >
specifies the matrices that you want to examine. The option `examine=I` displays the information matrix, $\mathbf{X}'\mathbf{X}$; `examine=V` displays the variance matrix, $(\mathbf{X}'\mathbf{X})^{-1}$; and `examine=I V` displays both. By default, these matrices are not displayed.

Specify `examine=aliasing=`*n* to examine the aliasing structure of the design. If you specify `examine= aliasing=2`, `%MktEx` will display the terms in the model and how they are aliased with up to two-factor interactions. More generally, with `examine=aliasing=`*n*, up to *n*-factor interactions are displayed. You can also specify `full` (e.g. `examine=aliasing=2 full`) to see the full (and often much more complicated) aliasing structure that PROC GLM produces directly. You can also specify `main` to see only the estimable functions that begin with main effects, and not the ones that begin with interactions. Interactions are still used with `examine=aliasing=2 main` and larger values of `aliasing=`. This option just removes some of the output.

Note that the `aliasing=`*n* option is resource intensive for larger problems. For some large problems, one of the underlying procedures might detect that the problem is too big, immediately issue an error, and quit. For other large problems, it might simply take a very long time before completing or printing an error due to insufficient resources. The number of two-way interaction terms is a quadratic function of the number of main effects, so it is not possible to print the aliasing structure even for some very reasonably sized main-effects designs.

interact= *interaction-list*
specifies interactions that must be estimable. By default, no interactions are guaranteed to be estimable. Examples:
`interact=x1*x2`
`interact=x1*x2 x3*x4*x5`
`interact=x1|x2|x3|x4|x5@2`
`interact=@2`

The interaction syntax is in most ways like PROC GLM's and many of the other modeling procedures. It uses "`*`" for simple interactions (`x1*x2` is the interaction between `x1` and `x2`), "`|`" for main effects and interactions (`x1|x2|x3` is the same as `x1 x2 x1*x2 x3 x1*x3 x2*x3 x1*x2*x3`) and "`@`" to eliminate higher-order interactions (`x1|x2|x3@2` eliminates `x1*x2*x3` and is the same as `x1 x2 x1*x2 x3 x1*x3 x2*x3`). The specification "`@2`" creates main effects and two-way interactions. Unlike PROC GLM's syntax, some short cuts are permitted. For the factor names, you can specify either the actual variable names (for example, `x1*x2` ...) or you can just specify the factor number without the "`x`" (for example, `1*2`). You can also specify `interact=@2` for all main effects and two-way interactions omitting the `1|2|`.... The following three specifications are equivalent:

```
   %mktex(2 ** 5, interact=@2, n=16)
   %mktex(2 ** 5, interact=1|2|3|4|5@2, n=16)
   %mktex(2 ** 5, interact=x1|x2|x3|x4|x5@2, n=16)
```

If you specify `interact=@2`, and if your specification matches a regular fractional-factorial design, then a resolution V design is requested. If instead you specify the full interaction list, (e.g. `interact=x1 | x2 | x3 | x4 | x5@2`) then the less-direct approach of requesting a design with the full list of interaction terms is taken, which in some cases might not work as well as directly requesting a resolution V design.

mintry= *n*

specifies the minimum number of rows to process before giving up for each design. For example, to ensure that the macro passes through each row of the design at least five times, you can specify `mintry=5 * n`. You can specify a number or a DATA step expression involving n (rows) and m (columns). By default, the macro will always consider at least n rows. This option can be useful with certain restrictions, particularly with `balance=`. When `balance=` and `mintry=mt` are both specified, then the balance restrictions are ignored for the first `mt - 3 * n / 2` passes through the design. During this period, the badness function for the balance restrictions is set to 1 so that `%MktEx` knows that the design does not conform. After that, all restrictions are considered. The `balance=` option works best when its restrictions are imposed on a reasonably efficient design not an inefficient initial design.

The `%MktEx` macro sometimes displays the following message:

```
WARNING: It may be impossible to meet all restrictions.
```

This message is displayed after `mintry=`*n* rows are passed without any success. Sometimes, it is premature to expect any success during the first pass. When you know this, you can specify this option to prevent that warning from coming out.

options= *options-list*

specifies binary options. By default, none of these options are specified. Specify one or more of the following values after `options=`.

`accept`
lets the macro output designs that violate restrictions imposed by `restrictions=`, `balance=`, or `partial=`, or have duplicates with `options=nodups`. Normally, the macro will not output such designs. With `options=accept`, a design becomes eligible for output when the macro can no longer improve on the restrictions or eliminate duplicates. Without `options=accept`, a design is only eligible when all restrictions are met and all duplicates are eliminated.

`check`
checks the efficiency of a given design, specified in `init=`, and disables the `out=`, `outr=`, and `outall=` options. If `init=` is not specified, `options=check` is ignored.

`file`
renders the design to a file with a generated file name. For example, if the design $2^{11}3^{12}$ in 36 runs is requested, the generated file name is: `OA(36,2^11,3^12)`. This option is ignored unless `options=render` is specified.

`int`
add an intercept to the design, variable, `x0`.

`justinit`
specifies that the macro should stop processing as soon as it is done making the initial design, even if that design would not normally be the final design. Usually, this design is an orthogonal array or some function of an orthogonal array (e.g. some three-level factors could be recoded into two-level factors), but there are no guarantees. Use this option when you want to output the initial design, for example, if you want to see the orthogonal but unbalanced design that `%MktEx` sometimes uses as an initial design. The `options=justinit` specification implies `optiter=0` and `outr=`. Also, `options=justinit nofinal` both stops the processing and prevents the final design from being evaluated. Particularly when you specify `options=nofinal`, you must ensure that this design has a suitable efficiency.

`largedesign`
lets the macro stop after `maxtime=` minutes have elapsed in the coordinate exchange algorithm. Typically, you would use this with `maxstages=1` and other options that make the algorithm run faster. By default, the macro checks time after it finishes with a design. With this option, the macro checks the time at the end of each row, after it has completed the first full pass through the design, and after any restrictions have been met, so the macro might stop before *D*-efficiency has converged. For really large problems and problems with restrictions, this option might make the macro run much faster but at a price of lower *D*-efficiency. For example, for large problems with restrictions, you might just want to try one run through the coordinate exchange algorithm with no candidate set search, orthogonal arrays, or mutations.

`lineage`
displays the lineage or "family tree" of the orthogonal array. For example, the lineage of the design $2^1 3^{25}$ in 54 runs is `54 ** 1 :  54 ** 1 > 3 ** 20 6 ** 1 9 ** 1 :  9 ** 1 > 3 ** 4 :  6 ** 1 > 2 ** 1 3 ** 1`. This states that the design starts as a single 54-level factor, then 54^1 is replaced by $3^{20} 6^1 9^1$, 9^1 is replaced by 3^4, and finally 6^1 is replaced by $2^1 3^1$ to make the final design.

`nodups`
eliminates duplicate runs.

`nofinal`
skips calling PROC OPTEX to display the efficiency of the final experimental design.

`nohistory`
does not display the iteration history.

`nooadups`
for orthogonal array construction, checks to see if a design with duplicate runs is created. If so, it tries using other factors from the larger orthogonal array to see if that helps avoid duplicates. There is no guarantee that this option will work. If you select only a small subset of the columns of an orthogonal array, duplicates might be unavoidable. If you really wish to ensure no duplicates, even at the expense of nonorthogonality, you must also specify `options=nodups`.

`noqc`
specifies that you do not have the SAS/QC product, so the `%MktEx` macro should try to get by without it. This means it tries to get by using the coordinate exchange and orthogonal array code without using PROC FACTEX and PROC OPTEX. The `%MktEx` macro will skip generating a candidate set, searching the candidate set, and displaying the final efficiency values. This is equivalent to specifying `optiter=0` and `options=nofinal`. This option also eliminates the check to see if the SAS/QC product is available. For some problems, the SAS/QC product is not necessary. For others, for example, for some orthogonal arrays in 128 runs, it is necessary. For other problems still, SAS/QC is not necessary, but the macro might find better designs if SAS/QC is available (for example, models with interactions and candidate sets on the order of a few thousand observations).

`nosort`
does not sort the design. One use of this option is with orthogonal arrays and Hadamard matrices. Some Hadamard matrices are generated with a banded structure that is lost when the design is sorted. If you want to see the original structure, and not just a design, specify `options=nosort`.

`nox`
suppresses the creation of `x1`, `x2`, `x3`, and so on, for use with the restrictions macro. If you are not using these names in your restrictions macro, specifying `options=nox` can make the macro run somewhat more efficiently. By default, `x1`, `x2`, `x3`, and so on are available for use.

`quick`
sets `optiter=0`, `maxdesigns=2`, `unbalanced=0`, and `tabiter=1`. This option provides a quick run that makes at most two design using coordinate exchange iterations—one using an initial design based on the orthogonal array table (catalog), and if necessary, one with a random initialization.

`quickr`
sets `optiter=0`, `maxdesigns=1`, `unbalanced=0`, and `tabiter=0`. This option provides an even quicker run than `options=quick` creating only one design using coordinate exchange and a random initialization. The "`r`" in `quickr` stands for random. You can use this option when you think using an initial design from the orthogonal array catalog will not help.

`quickt`
sets `optiter=0`, `maxdesigns=1`, `unbalanced=0`, and `tabiter=1`. This option provides an even quicker run than `options=quickr` with one design found by coordinate exchange using a design from the orthogonal array table (catalog) in the initialization. The "`t`" in `quickt` stands for table.

`render`
displays the design compactly in the SAS listing. If you specify `options=render file`, then the design is instead rendered to a file whose name represents the design specification. For example, if the design $2^{11}3^{12}$ in 36 runs is requested, the generated file name is: `OA(36,2^11,3^12)`.

`refine`
specifies that with an `init=` design data set with at least one nonpositive entry, each successive design iteration tries to refine the best design from before. By default, the part of the design that is not fixed is randomly reinitialized each time. The default strategy is usually superior.

`resrep`
reports on the progress of the restrictions. You should specify this option with problems with lots of restrictions. Always specify this option if you find that `%MktEx` is unable to make a design that conforms to the restrictions. By default, the iteration history is not displayed for the stage where `%MktEx` is trying to make the design conform to the restrictions. Specify `options=resrep` when you want to see the progress in making the design conform.

`+-`
with render, displays –1 as '–' and 1 as '+' in two-level factors. This option is typically used with `levels=i` for displaying Hadamard matrices.

`3`
modifies `options=+-` to apply to three-level factors as well: –1 as '–', 0 as '0', and 1 as '+'.

`512`
adds some larger designs in 512 runs with mixes of 16, 8, 4, and 2-level factors to the catalog, which gives added flexibility in 512 runs at a cost of potentially *much* slower run time. This option replaces the default $4^{160}32^{1}$ parent with $16^{32}32^{1}$ and adds over 60,000 new designs to the catalog. Many of these designs are automatically available with PROC FACTEX, so do not use this option unless you have first tried and failed to find the design without it.

partial= *n*

specifies a partial-profile design (Chrzan and Elrod 1995). The default is an ordinary linear model design. Specify, for example, `partial=4` if you only want 4 attributes to vary in each row of the design (except the first run, in which none vary). This option works by adding restrictions to the design (see `restrictions=`) and specifying `order=random` and `exchange=2`. The badness of each row (how far each row is from conforming to the partial-profile restrictions) is evaluated and the results stored in a scalar `__pbad`. When you specify other restrictions, this is added to the `bad` value created by your restrictions macro. You can use your restrictions macro to change or differentially weight `__pbad` before the final addition of the components of design badness takes place (see page 1065). Because of the default `exchange=2` with partial-profile designs, the construction is slow, so you might want to specify `maxdesigns=1` or other options to make `%MktEx` run faster. For large problems, you might get faster but less good results by specifying `order=seqran`. Specifying `options=accept` or `balance=` with `partial=` is *not* a good idea. The following steps create and display the first part of a partial-profile design with twelve factors, each of which has three levels that vary and one level that means the attribute is not shown:

```
%mktex(4 ** 12,                     /* 12 four-level factors              */
       n=48,                        /* 48 profiles                        */
       partial=4,                   /* four attrs vary                    */
       seed=205,                    /* random number seed                 */
       maxdesigns=1)                /* just make one design               */

%mktlab(data=randomized, values=. 1 2 3, nfill=99)

options missing=' ';
proc print data=final(obs=10); run;
options missing='.';
```

The first part of the design is as follows:

```
Obs    x1    x2    x3    x4    x5    x6    x7    x8    x9    x10    x11    x12

  1
  2                             3     1                 2             3
  3                             1           1     1            3
  4          2           2           1                                       3
  5    1                 1           3                                       2
  6    3           3                 1                                       2
  7          3                                          3             1      2
  8    2                             2           1                           3
  9                             3           1                         2      3
 10          3     1                 3           3
```

reslist= *list*

specifies a list of constant matrices. Begin all names with an underscore to ensure that they do not conflict with any of names that `%MktEx` uses. If you specify more than one name, then names must be separated by commas. Example: `reslist=%str(_a, _b)`.

resmac= *macro-name*

specifies the name of a macro that creates the matrices named in the `reslist=` option. Begin all names including all intermediate matrix names with an underscore to ensure that they do not conflict with any of the names that `%MktEx` uses.

The `reslist=` and `resmac=` options can be used jointly for certain complicated restrictions to set up some constant matrices that the restrictions macro can use. Since the restrictions macro is called a lot, anything you can do only once helps speed up the algorithm.

Another way you can use these options is when you want to access a `%MktEx` matrix in your restrictions macro that you normally could not access. This would require knowledge of the internal workings of the `%MktEx` macro, so it is not a capability that you would usually need.

restrictions= *macro-name*
specifies the name of a macro that places restrictions on the design. By default, there are no restrictions. If you have restrictions on the design, what combinations can appear with what other combinations, then you must create a macro that creates a variable called `bad` that contains a numerical summary of how bad the row of the design is. When everything is fine, set `bad` to zero. Otherwise set `bad` to a larger value that is a function of the number of restriction violations. The `bad` variable must not be binary (0 – ok, 1 – bad) unless there is only one simple restriction. You must set `bad` so that the `%MktEx` macro knows if the changes it is considering are moving the design in the right direction. See page 1079 for examples of restrictions. The macro must consist of PROC IML statements and possibly some macro statements.

When you have restrictions, you should usually specify `options=resrep` so that you can get a report on the restriction violations in the iteration history. This can be a great help in debugging your restrictions macro. Also, be sure to check the log when you specify `restrictions=`. The macro cannot always ensure that your statements are syntax-error free and stop if they are not. There are many options that can impose restrictions, including `restrictions=`, `options=nodups`, `balance=`, `partial=`, and `init=`. If you specify more than one of these options, be sure that the combination makes sense, and be sure that it is possible to simultaneously satisfy all of the restrictions.

Your macro can look at several scalars, along with a vector and a matrix in quantifying badness, and it must store its results in `bad`. The following names are available:

`i` – is a scalar that contains the number of the row currently being changed or evaluated. If you are writing restrictions that use the variable `i`, you almost certainly should specify `options=nosort`.

`try` – is a scalar similar to `i`, which contains the number of the row currently being changed. However, `try`, starts at zero and is incremented for each row, but it is only set back to zero when a new design starts, not when `%MktEx` reaches the last row. Use `i` as a matrix index and `try` to evaluate how far `%MktEx` is into the process of constructing the design.

`x` – is a row vector of factor levels for row `i` that always contains integer values beginning with 1 and continuing on to the number of levels for each factor. These values are always one-based, even if `levels=` is specified.

`x1` is the same as `x[1]`, `x2` is the same as `x[2]`, and so on.

`j1` – is a scalar that contains the number of the column currently being changed. In the steps where the badness macro is called once per row, `j1` = 1.

`j2` – is a scalar that contains the number of the other column currently being changed (along with `j1`) with `exchange=2`. Both `j1` and `j2` are defined when the `exchange=` value is greater than or equal to two. This scalar will not exist with `exchange=1`. In the steps where the badness macro is called once per row, `j1` = `j21` = 1.

`j3` – is a scalar that contains the number of the third column currently being changed (along with `j1` and `j2`) with `exchange=3` and larger `exchange=` values. This scalar will not exist with `exchange=1` and `exchange=2`. If and only if the `exchange=`value is greater than 3, there will be a `j4` and so on. In the steps where the badness macro is called once per row, `j1` = `j2` = `j3` = 1.

`xmat` – is the entire `x` matrix. Note that the *ith* row of `xmat` is often different from `x` since `x` contains information about the exchanges being considered, whereas `xmat` contains the current design.

`bad` – results: 0 – fine, or the number of violations of restrictions. You can make this value large or small, and you can use integers or real numbers. However, the values should always be nonnegative. When there are multiple sources of design badness, it is sometimes good to scale the different sources on different scales so that they do not trade off against each other. For example, for the first source, you might multiply the number of violations by 1000, by 100 for another source, by 10 for another source, by 1 for another source, and even sometimes by 0.1 or 0.01 for another source. The final badness is the sum of `bad`, `__pbad` (when it exists), and `__bbad` (when it exists). The scalars `__pbad` and `__bbad` are explained next.

`__pbad` – is the badness from the `partial=` option. When `partial=` is not specified, this scalar does not exist. Your macro can weight this value, typically by multiplying it times a constant, to differentially weight the contributors to badness, e.g.: `__pbad = __pbad * 10`.

`__bbad` – is the badness from the `balance=` option. When `balance=` is not specified, this scalar does not exist. Your macro can weight this value, typically by multiplying it times a constant, to differentially weight the contributors to badness, e.g.: `__bbad = __bbad * 100`.

Do not use these names (other than `bad`) for intermediate values!

Other than that, you can create intermediate variables without worrying about conflicts with the names in the macro. The levels of the factors for one row of the experimental design are stored in a vector `x`, and the first level is always 1, the second always 2, and so on. All restrictions must be defined in terms of `x[j]` (or alternatively, `x1`, `x2`, ..., and perhaps the other matrices). For example, if there are 5 three-level factors and if it is bad if the level of a factor equals the level for the following factor, you can create a macro `restrict` like the following and specify `restrictions=restrict` when you invoke the `%MktEx` macro:

```
%macro restrict;
   bad = (x1 = x2) +
         (x2 = x3) +
         (x3 = x4) +
         (x4 = x5);
   %mend;
```

Note that you specify just the macro name and no percents on the `restrictions=` option. Also note that IML does not have the full set of Boolean operators that the DATA step and other parts of SAS have. For example, these are *not* available: `OR AND NOT GT LT GE LE EQ NE`. Note that the expression `a <= b <= c` is perfectly valid in IML, but its meaning in IML is different than and less reasonable than its meaning in the DATA step. The DATA step expression checks to see if `b` is in the range of `a` to `c`. In contrast, the IML expression `a <= b <= c` is exactly the same as `(a <= b) <= c`, which evaluates `(a <= b)`, and sets the result to 0 (false) or 1 (true). Then IML compares the resulting 0 or 1 to see if it is less than or equal to `c`.

The operators you can use, along with their meanings, are as follows:

Specify	For	Do Not Specify
=	equals	EQ
∧ = or ¬ =	not equals	NE
<	less than	LT
<=	less than or equal to	LE
>	greater than	GT
>=	greater than or equal to	GE
&	and	AND
|	or	OR
∧ or ¬	not	NOT
`a` <= `b` & `b` <= `c`	range check	`a` <= `b` <= `c`

Restrictions can substantially slow down the algorithm.

With restrictions, the `Current D-Efficiency` column of the iteration history table might contain values larger than the `Best D-Efficiency` column. This is because the design corresponding to the current *D*-efficiency might have restriction violations. Values are only reported in the best *D*-efficiency column after all of the restriction violations have been removed. You can specify `options=accept` with `restrictions=` when it is okay if the restrictions are not met.

See page 1079 for more information about restrictions. See pages 471 and 604 for examples of restrictions. There are many examples of restrictions in the partial-profile examples starting on page 595.

seed= *n*

specifies the random number seed. By default, `seed=0`, and clock time is used to make the random number seed. By specifying a random number seed, results should be reproducible within a SAS release for a particular operating system and for a particular version of the macro. However, due to machine and macro differences, some results might not be exactly reproducible everywhere. For most orthogonal and balanced designs, the results should be reproducible. When computerized searches are done, it is likely that you will not get the same design across different computers, operating systems and different SAS and macro releases, although you would expect the efficiency differences to be slight.

Data Set Options

These next options specify the names of the input and output data sets.

cat= *SAS-data-set*

specifies the input design catalog. By default, the `%MktEx` macro automatically runs the `%MktOrth` macro to get this catalog. However, many designs can be made in multiple ways, so you can instead run `%MktOrth` yourself, select the exact design that you want, and specify the resulting data set in the `cat=` option. The catalog data set for input to `%MktEx` is the `outlev=` data set from the `%MktOrth` macro, which by default is called MKTDESLEV. Be sure to specify `options=dups lineage` when you run the `%MktOrth` macro. For example, the design 2^{71} in 72 runs can be made from either $2^{36}36^1$ or $2^{68}4^1$.

The following example shows how to select the $2^{36}36^1$ parent:

```
%mktorth(range=n=72, options=dups lineage)

proc print data=mktdeslev; var lineage; run;

data lev;
   set mktdeslev(where=(x2 = 71 and index(lineage, '2 ** 36 36 ** 1')));
   run;

%mktex(2 ** 71,                        /* 71 two-level factors              */
       n=72,                           /* 72 runs                           */
       cat=lev,                        /* OA catalog comes from lev data set   */
       out=b)                          /* output design                     */
```

The results of the following steps (not shown) show that you are in fact getting a design that is different from the default:

```
%mktex(2 ** 71, n=72, out=a)

proc compare data=a compare=b noprint note;
   run;
```

init= *SAS-data-set*

specifies the initial input experimental design. If all values in the initial design are positive, then a first step evaluates the design, the next step tries to improve it, and subsequent steps try to improve the best design found. However, if any values in the initial design are nonpositive (or missing) then a different approach is used. The initial design can have three types of values:

- positive integers are fixed and constant and will not change throughout the course of the iterations.
- zero and missing values are replaced by random values at the start of each new design search and can change throughout the course of the iterations.
- negative values are replaced by their absolute value at the start of each new design attempt and can change throughout the course of the iterations.

When absolute orthogonality and balance are required in a few factors, you can fix them in advance. The following steps illustrate how:

```
* Get first four factors;
%mktex(8 6 2 2, n=48)

* Flag the first four as fixed and set up to solve for the next six;
data init;
   set design;
   retain x5-x10 .;
   run;

* Get the last factors holding the first 4 fixed;
%mktex(8 6 2 2 4 ** 6,            /* append 4 ** 6 to 8 6 2 2            */
       n=48,                      /* 48 runs                             */
       init=init,                 /* initial design                      */
       maxiter=100)               /* 100 iterations                      */

%mkteval(data=design)
```

Alternatively, you can use `holdouts=` or `fixed=` to fix just certain rows.

out= *SAS-data-set*

specifies the output experimental design. The default is `out=Design`. By default, this design is sorted unless you specify `options=nosort`. This is the output data set to look at in evaluating the design. See the `outr=` option for a randomized version of the same design, which is usually more suitable for actual use. Specify a null value for `out=` if you do not want this data set created. Often, you will want to specify a two-level name to create a permanent SAS data set so the design is available later for analysis.

outall= *SAS-data-set*

specifies the output data set containing all designs found. By default, this data set is not created. This data set contains the design number, efficiency, and the design. If you use this option you can break while the macro is running and still recover the best design found so far. Note, however, that designs are only stored in this data set when they are done being processed. For example, design 1 is stored once at the end, not every time an improvement is found along the way. Make sure you store the data set in a permanent SAS data set. If, for example, your macro is currently working on the tenth design, with this option, you could break and get access to designs 1 through 9.

The following steps illustrate how you can use this option:

```
%mktex(2 2 2 3 3 3, n=18, outall=sasuser.a)

proc means data=sasuser.a noprint;
   output out=m max(efficiency)=m;
   run;

data best;
   retain id .;
   set sasuser.a;
   if _n_ eq 1 then set m;
   if nmiss(id) and abs(m - efficiency) < 1e-12 then do;
      id = design;
      put 'NOTE: Keeping design ' design +(-1) '.';
      end;
   if id eq design;
   keep design efficiency x:;
   run;

proc print; run;
```

outeff= *SAS-data-set*
specifies the output data set with the final efficiencies, the method used to find the design, and the initial random number seed. By default, this data set is not created.

outr= *SAS-data-set*
specifies the randomized output experimental design. The default is `outr=Randomized`. Levels are randomly reassigned within factors, and the runs are sorted into a random order. Neither of these operations affects efficiency. When `restrictions=` or `partial=` is specified, only the random sort is performed. Specify a null value for `outr=` if you do not want a randomized design created. Often, you will want to specify a two-level name to create a permanent SAS data set so the design is available later for analysis.

Iteration Options

These next options control some of the details of the iterations. The macro can perform three sets of iterations. The `Algorithm Search` set of iterations looks for efficient designs using three different approaches. It then determines which approach appears to be working best and uses that approach exclusively in the second set of `Design Search` iterations. The third set or `Design Refinement` iterations tries to refine the best design found so far by using level exchanges combined with random mutations and simulated annealing. Some of these options can take three arguments, one for each set of iterations.

The first set of iterations can have up to three parts. The first part uses either PROC PLAN or PROC FACTEX followed by PROC OPTEX, all called through the `%MktDes` macro, to create and search a candidate set for an optimal initial design. The second part might use an orthogonal array or fractional-factorial design as an initial design. The next part consists of level exchanges starting with random initial designs.

In the first part, if the full-factorial design is small and manageable (arbitrarily defined as < 5185 runs), it is used as a candidate set, otherwise a fractional-factorial candidate set is used. The macro tries `optiter=` iterations to make an optimal design using the `%MktDes` macro and PROC OPTEX.

In the second part, the macro tries to generate and improve a standard orthogonal array or fractional-factorial design. Sometimes, this can lead immediately to an optimal design, for example, with $2^{11}3^{12}$ and $n = 36$. In other cases, when only part of the desired design matches some standard design, only part of the design is initialized with the standard design and multiple iterations are run using the standard design as a partial initialization with the rest of the design randomly initialized.

In the third part, the macro uses the coordinate-exchange algorithm with random initial designs.

The following iteration options are available:

anneal= *n1 < n2 < n3 >>*

specifies the starting probability for simulated annealing in the coordinate-exchange algorithm. The default is `anneal=.05 .05 .01`. Specify a zero or null value for no annealing. You can specify more than one value if you would like to use a different value for the algorithm search, design search, and design refinement iterations. When you specify a value greater than zero and less than one, for example, 0.1 the design is permitted to get worse with decreasing probability as the number of iterations increases. This often helps the algorithm overcome local efficiency maxima. Permitting efficiency to decrease can help get past the bumps in the efficiency function.

Examples: `anneal=` or `anneal=0` specifies no annealing, `anneal=0.1` specifies an annealing probability of 0.1 during all three sets of iterations, `anneal=0 0.1 0.05` specifies no annealing during the initial iterations, an annealing probability of 0.1 during the search iterations, and an annealing probability of 0.05 during the refinement iterations.

anniter= *n1 < n2 < n3 >>*

specifies the first iteration to consider using annealing on the design. The default is `anniter=. . .`, which means that the macro chooses the values to use. The default is the first iteration that uses a fully random initial design in each of the three sets of iterations. Hence, by default, there is no random annealing in any part of the initial design when part of the initial design comes from an orthogonal design.

canditer= *n1 < n2 >*

specifies the number of coordinate-exchange iterations that are used to try to improve a candidate-set based, OPTEX-generated initial design. The default is `canditer=1 1`. Note that `optiter=` controls the number of OPTEX iterations. Unless you are using annealing or mutation in the `canditer=` iterations (by default you are not) or unless you are using `options=nodups`, do not change theses values. The default value of `canditer=1 1`, along with the default `mutiter=` and `anniter=` values of missing, mean that the results of the OPTEX iterations are presented once in the algorithm iteration history, and if appropriate, once in the design search iteration history. Furthermore, by default, OPTEX generated designs are not improved with level exchanges except in the design refinement phase.

maxdesigns= *n*

specifies that the macro should stop after `maxdesigns=` designs have been created. This option is useful with big, slow problems with restrictions. It is also useful as you are developing your design code. At first, just make one or a few designs, then when all of your code is finalized, let the macro run longer. You could specify, for example, `maxdesigns=3` and `maxtime=0` and the macro would perform one candidate-set-based iteration, one orthogonal design initialization iteration, and one random initialization iteration and then stop. By default, this option is ignored and stopping is based on the other iteration options. For large designs with restrictions, a typical specification is `options=largedesign quickr`, which is equivalent to `optiter=0, tabiter=0, unbalanced=0, maxdesigns=1, options=largedesign`.

maxiter= *n1 < n2 < n3 >>*
iter= *n1 < n2 < n3 >>*

specifies the maximum number of iterations or designs to generate. The default is `maxiter=21 25 10`. With larger values, the macro tends to find better designs at a cost of slower run times. You can specify more than one value if you would like to use a different value for the algorithm search, design search, and design refinement iterations. The second value is only used if the second set of iterations consists of coordinate-exchange iterations. Otherwise, the number of iterations for the second set is specified with the `tabiter=`, or `canditer=` and `optiter=` options. If you want more iterations, be sure to set the `maxtime=` option as well, because iteration stops when the maximum number of iterations is reached or the maximum amount of time, whichever comes first. Examples: `maxiter=10` specifies 10 iterations for the initial, search, and refinement iterations, and `maxiter=10 10 5` specifies 10 initial iterations, followed by 10 search iterations, followed by 5 refinement iterations.

maxstages= *n*

specifies that the macro should stop after `maxstages=` algorithm stages have been completed. This option is useful for big designs, and for times when the macro runs slowly, for example, with restrictions. You could specify `maxstages=1` and the macro will stop after the algorithm search stage, or `maxstages=2` and the macro will stop after the design search stage. The default is `maxstages=3`, which means the macro will stop after the design refinement stage.

maxtime= *n1 < n2 < n3 >>*

specifies the approximate maximum amount of time in minutes to run each phase. The default is `maxtime=10 20 5`. When an iteration completes (a design is completed), if more than the specified amount of time has elapsed, the macro quits iterating in that phase. Usually, run time is no more than 10% or 20% larger than the specified values. However, for large problems, with restrictions, and with `exchange=` values other than 1, run time might be quite a bit larger than the specified value, since the macro only checks time after a design finishes.

You can specify more than one value if you would like to use a different value for the algorithm search, design search, and design refinement iterations. By default, the macro spends up to 10 minutes on the algorithm search iterations, 20 minutes on the design search iterations, and 5 minutes in the refinement stage. Most problems run in much less time than this. Note that the second value is ignored for OPTEX iterations since OPTEX does not have any timing options. This option also affects, in the algorithm search iterations, when the macro switches between using an orthogonal initial design to using a random initial design. If the macro is not done using orthogonal initializations, and one half of the first time value has passed, it switches. Examples: `maxtime=60` specifies up to one hour for each phase. `maxtime=20 30 10` specifies 20 minutes for the first phase and 30 minutes for the second, and 10 for the third.

The option `maxtime=0` provides a way to get a quick run, with no more than one iteration in each phase. However, even with `maxtime=0`, run time can be several minutes or more for large problems. See the `maxdesigns=` and `maxstages=` options for other ways to drastically cut run time for large problems. If you specify really large time values (anything more than hours), you probably need to also specify `optiter=` since the default values depend on `maxtime=`.

mutate= *n1 < n2 < n3 >>*

specifies the probability at which each value in an initial design can mutate or be assigned a different random value before the coordinate-exchange iterations begin. The default is `mutate=.05 .05 .01`. Specify a zero or null value for no mutation. You can specify more than one value if you would like to use a different value for the algorithm search, design search, and design refinement iterations. Examples: `mutate=` or `mutate=0` specifies no random mutations. The `mutate=0.1` option specifies a mutation probability of 0.1 during all three sets of iterations. The `mutate=0 0.1 0.05` option specifies no mutations during the first iterations, a mutation probability of 0.1 during the search iterations, and a mutation probability of 0.05 during the refinement iterations.

mutiter= *n1 < n2 < n3 >>*

specifies the first iteration to consider mutating the design. The default is `mutiter=. . .`, which means that the macro chooses values to use. The default is the first iteration that uses a fully random initial design in each of the three sets of iterations. Hence, by default, there are no random mutations of any part of the initial design when part of the initial design comes from an orthogonal design.

optiter= *n1 < n2 >*

specifies the number of iterations to use in the OPTEX candidate-set based searches in the algorithm and design search iterations. The default is `optiter=. .`, which means that the macro chooses values to use. When the first value is "`.`" (missing), the macro will choose a value usually no smaller than 20 for larger problems and usually no larger than 200 for smaller problems. However, `maxtime=` values other than the defaults can make the macro choose values outside this range. When the second value is missing, the macro will choose a value based on how long the first OPTEX run took and the value of `maxtime=`, but no larger than 5000. When a missing value is specified for the first `optiter=` value, the default, the macro might choose to not perform any OPTEX iterations to save time if it thinks it can find a perfect design without them.

repeat= *n1 n2 n3*

specifies the maximum number of times to repeatedly work on a row to eliminate restriction violations. The default value of `repeat=25 . .` specifies that a row should be worked on up to 25 times to eliminate violations. The second value is the place in the design refinement where this processing starts. This is based on a zero-based total number of rows processed so far. This is like a zero-based row index, but it never resets within a design. The third value is the place where this extra repeated processing stops. Let m be the `mintry=`m value, which by default is n, the number of rows. By default, when the second value is missing, the process starts after m rows have been processed (the second complete pass through the design). By default, the process stops after `m + 10 * n` rows have been processed where m is the second (specified or derived) `repeat=` value.

tabiter= *n1 < n2 >*

specifies the number of times to try to improve an orthogonal or fractional-factorial initial design. The default is `tabiter=10 200`, which means 10 iterations in the algorithm search and 200 iterations in the design search.

unbalanced= *n1 < n2 >*

specifies the proportion of the `tabiter=` iterations to consider using unbalanced factors in the initial design. The default is `unbalanced=.2 .1`. One way that unbalanced factors occur is through coding down. Coding down, for example, creates a three-level factor from a four-level factor: (1 2 3 4) $\Rightarrow$ (1 2 3 3) or a two-level factor from a three-level factor: (1 2 3) $\Rightarrow$ (1 2 2). For any particular problem, this strategy is probably either going to work really well or not well at all, without much variability in the results, so it is not tried very often by default. This option will try to create two-level through five-level factors from three-level through six-level factors. It will not attempt, for example, to code down a twenty-level factor into a nineteen-level factor (although the macro is often capable of in effect doing just that through level exchanges).

In this problem, for example, the optimal design is constructed by coding down a six-level factor into a five-level factor:

```
%mktex(3 3 3 3 5, n=18)

%mkteval;
```

The frequencies for the levels of the five-level factor are 6, 3, 3, 3, and 3. The optimal design is orthogonal but unbalanced.

Miscellaneous Options

This section contains some miscellaneous options that some users might occasionally find useful.

big= *n* < `choose` >

specifies the full-factorial-design size that is considered to be big. The default is `big=5185 choose`. The default value was chosen because 5185 is approximately 5000 and greater than $2^6 3^4 = 5184$, $2^{12} = 4096$, and $2 \times 3^7 = 4374$. When the full-factorial design is smaller than the `big=` value, the `%MktEx` macro searches a full-factorial candidate set. Otherwise, it searches a fractional-factorial candidate set. When `choose` is specified as well (the default), the macro can choose to use a fractional-factorial even if the full-factorial design is not too big if it appears that the final design can be created from the fractional-factorial design. This might be useful, for example, when you are requesting a fractional-factorial design with interactions. Using FACTEX to create the fractional-factorial design might be a better strategy than searching a full-factorial design with PROC OPTEX.

exchange= *n*

specifies the number of factors to consider at a time when exchanging levels. You can specify `exchange=2` to do pairwise exchanges. Pairwise exchanges are *much* slower, but they might produce better designs. For this reason, you might want to specify `maxtime=0` or `maxdesigns=1` or other iteration options to make fewer designs and make the macro run faster. The `exchange=` option interacts with the `order=` option. The `order=seqran` option is faster with `exchange=2` than `order=sequential` or `order=random`. The default is `exchange=2` when `partial=` is specified. With `order=matrix`, the `exchange=` value is the number of matrix columns. Otherwise, the default is `exchange=1`.

With partial-profile designs and certain other highly restricted designs, it is important to do pairwise exchanges. Consider, for example, the following design row with `partial=4`:

```
    1 1 2 3 1 1 1 2 1 1 1 3
```

The `%MktEx` macro cannot consider changing a 1 to a 2 or 3 unless it can also consider changing one of the current 2's or 3's to 1 to maintain the partial-profile restriction of exactly four values not equal to 1. Specifying the `exchange=2` option gives `%MktEx` that flexibility.

fixed= *variable*

specifies an `init=` data set variable that indicates which runs are fixed (cannot be changed) and which ones can be changed. By default, no runs are fixed.

1 – (or any nonmissing) means this run must never change.
0 – means this run is used in the initial design, but it can be swapped out.
. – means this run should be randomly initialized, and it can be swapped out.

This option can be used to add holdout runs to a conjoint design, but see `holdouts=` for an easier way. To fix parts of the design in a much more general way, see the `init=` option.

holdouts= *n*

adds holdout observations to the `init=` data set. This option augments an initial design. Specifying `holdouts=`*n* optimally adds *n* runs to the `init=` design. The option `holdouts=`*n* works by adding a `fixed=` variable and extra runs to the `init=` data set. Do not specify both `fixed=` and `holdouts=`. The number of rows in the `init=` design, plus the value specified in `holdouts=` must equal the `n=` value.

levels= *value*

specifies the method of assigning the final factor levels. This recoding occurs after the design is created, so all restrictions must be expressed in terms of one-based factors, regardless of what is specified in the `levels=` option.

Values:

`1` – default, one based, the levels are 1, 2, ...

`0` – zero based, the levels are 0, 1, ...

`c` – centered, possibly resulting in nonintegers 1 2 → –0.5 0.5, 1 2 3 → –1 0 1.

`i` – centered and scaled to integers. 1 2 → –1 1, 1 2 3 → –1 0 1.

You can also specify separate values for two- and three-level factors by preceding a value by "2" or "3". For example, `levels=2 i 3 0 c` means two-level factors are coded –1, 1 and three-level factors are coded 0, 1, 2. The remaining factors are centered. Note that the centering is based on centering the level values not on centering the (potentially unbalanced) factor. So, for example, the centered levels for a two-level factor in five runs (1 2 1 2 1) are (–0.5 0.5 –0.5 0.5 –0.5) not (–0.4 0.6 –0.4 0.6 –0.4). If you want the latter form of centering, use `proc standard m=0`. See the `%MktLab` macro for more general level setting.

You can also specify three other values:

`first` – means the first row of the design should consist entirely of the first level.

`last` – means the first row of the design should consist entirely of the last level, which is useful for Hadamard matrices.

`int` – adds an intercept column to the design.

order= `col=`*n* | `matrix=SAS-data-set`| `random` | `random=`*n* | `ranseq` | `sequential`
specifies the order in which the columns are worked on in the coordinate exchange algorithm. Valid values include:

`col=`*n* – process *n* random columns in each row
`matrix=`*SAS-data-set* – read the order from a data set
`random` – random order
`random=`*n* – random order with partial-profile exchanges
`ranseq` – sequential from a random first column
`seqran` – alias for `ranseq`
`sequential` – 1, 2, 3, ...
null – (the default) `random` when there are partial-profile restrictions, `ranseq` when there are other restrictions, and `sequential` otherwise.

For `order=col=`*n*, specify an integer for *n*, for example, `order=col=2`. This option should only be used for huge problems where you do not care if you hit every column. Typically, this option is used in conjunction with `options=largedesign quickr`. You would use it when you have a large problem and you do not have enough time for one complete pass through the design. You just want to iterate for approximately the `maxtime=` amount of time then stop. You should not use `order=col=` with restrictions.

The options `order=random=`*n* is like `order=random`, but with an adaptation that is particularly useful for partial-profile choice designs. Use this option with `exchange=2`. Say you are making a partial-profile design with ten attributes and three alternatives. Then attribute 1 is made from `x1`, `x11`, and `x21`; attribute 2 is made from `x2`, `x12`, and `x22`; and so on. Specifying `order=random=10` means that the columns, as shown by column index `j1`, are traversed in a random order. A second loop (with variable `j2`) traverses all of the factors in the current attribute. So, for example, when `j1` is 13, then `j2` = 3, 13, 23. This performs pairwise exchanges within choice attributes.

The `order=` option interacts with the `exchange=` option. With a random order and `exchange=2`, the variable `j1` loops over the columns of the design in a random order and for each `j1`, `j2` loops over the columns greater than `j1` in a random order. With a sequential order and `exchange=2`, the variable `j1` loops over the columns in 1, 2, 3 order and for each `j1`, `j2` loops over the columns greater than `j1` in a `j1`+1, `j1`+2, `j1`+3 order. The `order=ranseq` option is different. With `exchange=2`, the variable `j1` loops over the columns in an order r, $r+1$, $r+2$, ..., m, 1, 2, ..., $r-1$ (for random r), and for each `j1` there is a single random `j2`. Hence, `order=ranseq` is the fastest option since it does not consider all pairs, just one pair. The `order=ranseq` option provides the only situation where you might try `exchange=3`.

The `order=matrix=SAS-data-set` option lets you specify exactly what columns are worked on, in what order, and in what groups. The SAS data set provides one row for every column grouping. Say you want to use this option to work on columns in pairs. (Note that you could just use `exchange=2` to do this.) Then the data set would have two columns. The first variable contains the number of a design column, and the second variable contains the number of a second column that is to be exchanged with the first. The names of the variables are arbitrary. The following steps create and display an example data set for five factors:

```
%let m = 5;
data ex;
   do i = 1 to &m;
      do j = i + 1 to &m;
         output;
         end;
      end;
   run;

proc print noobs; run;
```

The results are as follows:

```
i    j

1    2
1    3
1    4
1    5
2    3
2    4
2    5
3    4
3    5
4    5
```

The specified `exchange=` value is ignored, and the actual `exchange=` value is set to two because the data set has two columns. The values must be integers between 1 and m, where m is the number of factors. The values can also be missing except in the first column. Missing values are replaced by a random column (potentially a different random column each time).

In a model with interactions, you can use this option to ensure that the terms that enter into interactions together get processed together. This is illustrated in the following steps:

```
data mat;
   input x1-x3;
   datalines;
1 1 1
2 3 .
2 4 .
3 4 .
2 3 4
5 5 .
6 7 .
8 . .
;

%mktex(4 4 2 2 3 3 2 3,             /* levels of all the factors          */
       n=36,                        /* 36 runs                            */
       order=matrix=mat,            /* matrix of columns to work on       */
       interact=x2*x3 x2*x4 x3*x4 x6*x7, /* interactions                  */
       seed=472)                    /* random number seed                 */
```

The data set MAT contains eight rows, so there are eight column groupings processed. The data set contains three columns, so up to three-way exchanges are considered. The first row mentions column 1 three times. Any repeats of a column number are ignored, so the first group of columns simply consists of column 1. The second column consists of 2, 3, and ., so the second group consists of columns 2, 3, and some random column. The random column could be any of the columns including 2 and 3, so this will sometimes be a two-way and sometimes be a three-way exchange. This group was specified since `x2*x3` is one of the interaction terms. Similarly, other groups consist of the other two-way interaction terms and a random factor: 2 and 4, 3 and 4, and 6 and 7. In addition, to help with the 3 two-way interactions involving `x2`, `x3`, and `x4`, there is one three-way term. Each time, this will consider $4 \times 2 \times 2$ exchanges, the product of the three numbers of levels. In principle, there is no limit on the number of columns, but in practice, this number could easily get too big to be useful with more than a few exchanges at a time. The row `5 5 .` requests an exchange between column 5 and a random factor. The row `8 .  .` requests an exchange between column 8 and two random factors.

stopearly= *n*

specifies that the macro can stop early when it keeps finding the same maximum *D*-efficiency over and over again in different designs. The default is `stopearly=5`. By default, during the design search iterations and refinement iterations, the macro will stop early if 5 times, the macro finds a *D*-efficiency essentially equal to the maximum but not greater than the maximum. This might mean that the macro has found the optimal design, or it might mean that the macro keeps finding a very attractive local optimum. Either way, it is unlikely it will do any better. When the macro stops for this reason, the macro will display the following message:

```
NOTE: Stopping since it appears that no improvement is possible.
```

Specify either 0 or a very large value to turn off the stop-early checking.

tabsize= *n*
provides you with some control on which design (orthogonal array, FACTEX or Hadamard) from the orthogonal array table (catalog) is used for the partial initialization when an exact match is not found. Specify the number of runs in the orthogonal array. By default, the macro chooses an orthogonal design that best matches the specified design. See the `cat=` option for more detailed control.

target= *n*
specifies the target efficiency criterion. The default is `target=100`. The macro stops when it finds an efficiency value greater than or equal to this number. If you know what the maximum efficiency criterion is, or you know how big is big enough, you can sometimes make the macro run faster by letting it stop when it reaches the specified efficiency. You can also use this option if you just want to see the initial design that `%MktEx` is using: `target=1, optiter=0`. By specifying `target=1`, the macro will stop after the initialization as long as the initial efficiency is ≥ 1.

Esoteric Options

This last set of options contains all of the other miscellaneous options. Most of the time, most users should not specify options from this list.

annealfun= *function*
specifies the function that controls how the simulated annealing probability changes with each pass through the design. The default is `annealfun=anneal * 0.85`. Note that the IML operator `#` performs ordinary (scalar) multiplication. Most users will never need this option.

detfuzz= *n*
specifies the value used to determine if determinants are changing. The default is `detfuzz=1e-8`. If `newdeter > olddeter * (1 + detfuzz)` then the new determinant is larger. Otherwise if `newdeter > olddeter * (1 - detfuzz)` then the new determinant is the same. Otherwise the new determinant is smaller. Most users will never need this option.

imlopts= *options*
specifies IML PROC statement options. For example, for very large problems, you can use this option to specify the IML `symsize=` or `worksize=` options: `imlopts=symsize=`*n* `worksize=`*m*, substituting numeric values for *n* and *m*. The defaults for these options are host dependent. Most users will never need this option.

ridge= *n*
specifies the value to add to the diagonal of $\mathbf{X}'\mathbf{X}$ to make it nonsingular. The default is `ridge=1e-7`. Usually, for normal problems, you will not need to change this value. If you want the macro to create designs with more parameters than runs, you must specify some other value, usually something like 0.01. By default, the macro will quit when there are more parameters than runs. Specifying a `ridge=` value other than the default (even if you just change the "e" in 1e–7 to "E") lets the macro create a design with more parameters than runs. This option is sometimes needed for advanced design problems.

Advanced Restrictions

It is extremely important with restrictions to appropriately quantify the badness of the run. The `%MktEx` macro has to know when it considers an exchange if it is considering:

- eliminating restriction violations making the design better,
- causing more restriction violations making the design worse,
- a change that neither increases nor decreases the number of violations.

Your restrictions macro must tell `%MktEx` when it is making progress in the right direction. If it does not, `%MktEx` will probably not find an acceptable design.

Complicated Restrictions

Consider designing a choice experiment with two alternatives each composed of 25 attributes, and the first 22 of which have restrictions on them. Attribute one in the choice design is made from `x1` and `x23`, attribute two in the choice design is made from `x2` and `x24`, ..., and attribute 22 in the choice design is made from `x22` and `x44`. The remaining attributes are made from `x45 -- x50`. The restrictions are as follows: each choice attribute must contain two 1's between 5 and 9 times, each choice attribute must contain exactly one 1 between 5 and 9 times, and each choice attribute must contain two 2's between 5 and 9 times. The following steps show an example of how *NOT* to accomplish this:

```
%macro sumres;
   allone = 0; oneone = 0; alltwo = 0;
   do k = 1 to 22;
      if      (x[k] = 1 & x[k+22] = 1) then allone = allone + 1;
      else if (x[k] = 1 | x[k+22] = 1) then oneone = oneone + 1;
      else if (x[k] = 2 & x[k+22] = 2) then alltwo = alltwo + 1;
      end;

   * Bad example.  Need to quantify badness.;
   bad = (^((5 <= allone & allone <= 9) &
            (5 <= oneone & oneone <= 9) &
            (5 <= alltwo & alltwo <= 9)));
   %mend;

%mktex(3 ** 50,                    /* 50 three-level factors               */
       n=135,                      /* 135 runs                             */
       restrictions=sumres,        /* name of restrictions macro           */
       seed=289,                   /* random number seed                   */
       options=resrep              /* restrictions report                  */
               quickr              /* very quick run with random init      */
               nox)                /* suppresses x1, x2, x3 ... creation   */
```

The problem with the preceding approach is there are complicated restrictions but badness is binary. If all the counts are in the right range, badness is 0, otherwise it is 1. You need to write a macro that lets `%MktEx` know when it is going in the right direction or it will probably never find a suitable design. One thing that is correct about the preceding code is the compound Boolean range expressions like `(5`

`<= allone & allone <= 9)`. Abbreviated expressions like `(5 <= allone <= 9)` that work correctly in the DATA step work incorrectly and without warning in IML. Another thing that is correct is the way the `sumres` macro creates new variables, `k`, `allone`, `oneone`, and `alltwo`. Care was taken to avoid using names like `i` and `x` that conflict with the matrices that you can examine in quantifying badness. The full list of names that you must avoid are `i`, `try`, `x`, `x1`, `x2`, ..., through `x`*n* for *n* factors, `j1`, `j2`, `j3`, and `xmat`. The following steps show a slightly better but still bad example of the macro:

```
%macro sumres;
   allone = 0; oneone = 0; alltwo = 0;
   do k = 1 to 22;
      if      (x[k] = 1 & x[k+22] = 1) then allone = allone + 1;
      else if (x[k] = 1 | x[k+22] = 1) then oneone = oneone + 1;
      else if (x[k] = 2 & x[k+22] = 2) then alltwo = alltwo + 1;
      end;
   * Better, badness is quantified, and almost correctly too!;
   bad = (^((5 <= allone & allone <= 9) &
            (5 <= oneone & oneone <= 9) &
            (5 <= alltwo & alltwo <= 9))) #
         (abs(allone - 7) + abs(oneone - 7) + abs(alltwo - 7));
   %mend;

%mktex(3 ** 50,                    /* 50 three-level factors            */
       n=135,                      /* 135 runs                          */
       restrictions=sumres,        /* name of restrictions macro        */
       seed=289,                   /* random number seed                */
       options=resrep              /* restrictions report               */
               quickr              /* very quick run with random init   */
               nox)                /* suppresses x1, x2, x3 ... creation */
```

This restrictions macro seems at first glance to do everything right—it quantifies badness. We need to examine this macro more closely. It counts in `allone`, `oneone`, and `alltwo` the number of times choice attributes are all one, have exactly one 1, or are all two. Everything is fine when the all one count is in the range 5 to 9 `(5 <= allone & allone <= 9)`, and the exactly one 1 count is in the range 5 to 9 `(5 <= oneone & oneone <= 9)`, and the all two count is in the range 5 to 9 `(5 <= alltwo & alltwo <= 9)`. It is bad when this is not true `(^((5 <= allone & allone <= 9) & (5 <= oneone & oneone <= 9) & (5 <= alltwo & alltwo <= 9)))`, the Boolean not operator "^" performs the logical negation. This Boolean expression is 1 for bad and 0 for OK. It is multiplied times a quantitative sum of how far these counts are outside the right range `(abs(allone - 7) + abs(oneone - 7) + abs(alltwo - 7))`. When the run meets all restrictions, this sum of absolute differences is multiplied by zero. Otherwise badness gets larger as the counts get farther away from the middle of the 5 to 9 interval.

In the `%MktEx` macro, we specify `options=resrep` to produce a report in the iteration history on the process of meeting the restrictions. When you run `%MktEx` and it is having trouble making a design that conforms to restrictions, this report can be extremely helpful. Next, we will examine some of the output from running the preceding macros.

Some of the results are as follows:

```
                          Algorithm Search History

                                Current         Best
       Design    Row,Col  D-Efficiency  D-Efficiency  Notes
       ----------------------------------------------------------
            1      Start       59.7632                Ran,Mut,Ann
            1      1           60.0363                0 Violations
            1      2           60.3715                0 Violations
            1      3           60.9507                0 Violations
            1      4           61.2319                5 Violations
            1      5           61.6829                0 Violations
            1      6           62.1529                0 Violations
            1      7           62.4004                0 Violations
            1      8           62.9747                3 Violations
            .
            .
            .

            1    132           70.4482                6 Violations
            1    133           70.3394                4 Violations
            1    134           70.4054                0 Violations
            1    135           70.4598                0 Violations
```

So far we have seen the results from the first pass through the design. With `options=resrep` the macro displays one line per row with the number of violations when it is done with the row. Notice that the macro is succeeding in eliminating violations in some but not all rows. This is the first thing you should look for. If it is not succeeding in any rows, you might have written a set of restrictions that is impossible to satisfy. Some of the output from the second pass through the design is as follows:

```
            1      1           70.5586                0 Violations
            1      2           70.7439                0 Violations
            1      3           70.7383                0 Violations
            1      4           70.7429                5 Violations
            1      4           70.6392                4 Violations
            1      4           70.7081                4 Violations
            1      4           70.7717                4 Violations
            1      4           70.7717                4 Violations
            1      4           70.7717                4 Violations
            1      4           70.7717                4 Violations
            1      4           70.7717                4 Violations
            1      4           70.7202                4 Violations
            1      4           70.7717                4 Violations
            1      4           70.7717                4 Violations
            1      4           70.7717                4 Violations
```

```
           1      4           70.7264                4 Violations
           1      4           70.7717                4 Violations
           1      4           70.7717                4 Violations
           1      4           70.7717                4 Violations
           1      4           70.7717                4 Violations
           1      4           70.7717                4 Violations
           1      4           70.7274                4 Violations
           1      4           70.7717                4 Violations
           1      4           70.7515                4 Violations
           1      4           70.7636                4 Violations
           1      4           70.7717                4 Violations
           1      4           70.7591                4 Violations
           1      4           70.7717                4 Violations
           1      5           70.7913                0 Violations
           1      6           70.9467                0 Violations
           1      7           71.0102                0 Violations
           1      8           71.0660                0 Violations
```

In the second pass, in situations where the macro had some reasonable success in the first pass, `%MktEx` tries extra hard to impose restrictions. We see it trying over and over again without success to impose the restrictions in the fourth row. All it manages to do is lower the number of violations from 5 to 4. We also see it has no trouble removing all violations in the eighth row that were still there after the first pass. The macro produces volumes of output like this. For several iterations, it will devote extra attention to rows with some violations but in this case without complete success. When you see this pattern, some success but also some stubborn rows that the macro cannot fix, there might be something wrong with your restrictions macro. Are you *really* telling `%MktEx` when it is doing a better job? These preceding steps illustrate some of the things that can go wrong with restrictions macros. It is important to carefully evaluate the results—look at the design, look at the iteration history, specify `options=resrep`, and so on to ensure your restrictions are doing what you want. The problem in this case is in the quantification of badness, in the following statement:

```
   bad = (^((5 <= allone & allone <= 9) &
            (5 <= oneone & oneone <= 9) &
            (5 <= alltwo & alltwo <= 9))) #
         (abs(allone - 7) + abs(oneone - 7) + abs(alltwo - 7));
```

Notice that we have three nonindependent contributors to the badness function, the three counts. As a level gets changed, it could increase one count and decrease another. There is a larger problem too. Say that `allone` and `oneone` are in the right range but `alltwo` is not. Then the function fragments `abs(allone - 7)` and `abs(oneone - 7)` incorrectly contribute to the badness function. The fix is to clearly differentiate the three sources of badness *and* weight the pieces so that one part never trades off against the other, for example, as follows:

```
%macro sumres;
   allone = 0; oneone = 0; alltwo = 0;
   do k = 1 to 22;
      if      (x[k] = 1 & x[k+22] = 1) then allone = allone + 1;
      else if (x[k] = 1 | x[k+22] = 1) then oneone = oneone + 1;
      else if (x[k] = 2 & x[k+22] = 2) then alltwo = alltwo + 1;
      end;
   bad = 100 # (^(5 <= allone & allone <= 9)) # abs(allone - 7) +
          10 # (^(5 <= oneone & oneone <= 9)) # abs(oneone - 7) +
               (^(5 <= alltwo & alltwo <= 9)) # abs(alltwo - 7);
   %mend;

%mktex(3 ** 50,                    /* 50 three-level factors             */
       n=135,                      /* 135 runs                           */
       restrictions=sumres,        /* name of restrictions macro         */
       seed=289,                   /* random number seed                 */
       options=resrep              /* restrictions report                */
               quickr              /* very quick run with random init    */
               nox)                /* suppresses x1, x2, x3 ... creation */
```

Now a component of the badness only contributes to the function when it is really part of the problem. We gave the first part weight 100 and the second part weight 10. Now the macro will never change `oneone` or `alltwo` if that causes a problem for `allone`, and it will never change `alltwo` if that causes a problem for `oneone`. Previously, the macro was getting stuck in some rows because it could never figure out how to fix one component of badness without making another component worse. *For some problems, figuring out how to differentially weight the components of badness so that they never trade off against each other is the key to writing a successful restrictions macro.* Often, it does not matter which component gets the most weight. What is important is that each component gets a *different* weight so that `%MktEx` does not get caught cycling back and forth making A better and B worse then making B better and A worse. Some of the output from the first pass through the design is as follows:

```
                         Algorithm Search History

                              Current         Best
Design    Row,Col  D-Efficiency  D-Efficiency  Notes
----------------------------------------------------------
     1      Start       59.7632                Ran,Mut,Ann
     1      1           60.1415                0 Violations
     1      2           60.5303                0 Violations
     1      3           61.0148                0 Violations
     1      4           61.4507                0 Violations
     1      5           61.7717                0 Violations
     1      6           62.2353                0 Violations
     1      7           62.5967                0 Violations
     1      8           63.1628                3 Violations

     .
     .
     .
```

```
               1    126           72.3566                   4 Violations
               1    127           72.2597                   0 Violations
               1    128           72.3067                   0 Violations
               1    129           72.3092                   0 Violations
               1    130           72.0980                   0 Violations
               1    131           71.8163                   0 Violations
               1    132           71.3795                   0 Violations
               1    133           71.4446                   0 Violations
               1    134           71.2805                   0 Violations
               1    135           71.3253                   0 Violations
```

We can see that in the first pass, the macro is imposing all restrictions for most but not all of the rows. Some of the output from the second pass through the design is as follows:

```
               1      1           71.3968                   0 Violations
               1      2           71.5017                   0 Violations
               1      3           71.7295                   0 Violations
               1      4           71.7839                   0 Violations
               1      5           71.8671                   0 Violations
               1      6           71.9544                   0 Violations
               1      7           72.0444                   0 Violations
               1      8           72.0472                   0 Violations
               .
               .
               .

               1    126           77.1597                   0 Violations
               1    127           77.1604                   0 Violations
               1    128           77.1323                   0 Violations
               1    129           77.1584                   0 Violations
               1    130           77.0708                   0 Violations
               1    131           77.1013                   0 Violations
               1    132           77.1721                   0 Violations
               1    133           77.1651                   0 Violations
               1    134           77.1651                   0 Violations
               1    135           77.2061                   0 Violations
```

In the second pass, `%MktEx` has imposed all the restrictions in rows 8 and 126, the rows that still had violations after the first pass (and all of the other not shown rows too). The third pass ends with the following output:

```
             1    126          78.7813                  0 Violations
             1    127   1      78.7813       78.7813    Conforms
             1    127  18      78.7899       78.7899
             1    127  19      78.7923       78.7923
             1    127  32      78.7933       78.7933
             1    127  40      78.7971       78.7971
             1    127  44      78.8042       78.8042
             1    127  47      78.8250       78.8250
             1    127  50      78.8259       78.8259
             1    127   1      78.8296       78.8296
             1    127   5      78.8296       78.8296
             1    127   8      78.8449       78.8449
             1    127  10      78.8456       78.8456
             1    128  48      78.8585       78.8585
             1    128  49      78.8591       78.8591
             1    128   7      78.8591       78.8591
```

The `%MktEx` macro completes a full pass through row 126, the place of the last violation, without finding any new violations so the macro states in row 127 that the design conforms to the restrictions and the iteration history proceeds in the normal fashion from then on (not shown). The note `Conforms` is displayed at the place where `%MktEx` decides that the design conforms. The design will continue to conform throughout more iterations, even though the note `Conforms` is not displayed on every line. The final efficiency is as follows:

```
                              The OPTEX Procedure

                                                                  Average
                                                                 Prediction
   Design                                                         Standard
   Number     D-Efficiency     A-Efficiency     G-Efficiency        Error
   ------------------------------------------------------------------------
        1          85.0645          72.2858          95.6858         0.8650
```

These next steps create the choice design and display a subset of the design:

```
%mktkey(x1-x50)

data key;
   input (x1-x25) ($);
   datalines;
x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13
x14 x15 x16 x17 x18 x19 x20 x21 x22                  x45 x46 x47
x23 x24 x25 x26 x27 x28 x29 x30 x31 x32 x33 x34
x35 x36 x37 x38 x39 x40 x41 x42 x43 x44              x48 x49 x50
;
```

```
%mktroll(design=design, key=key, out=chdes)

proc print; by set; id set; where set le 2 or set ge 134; run;
```

Notice the slightly unusual arrangement of the `Key` data set due to the fact that the first 22 attributes get made from the first 44 factors of the linear arrangement.

The first four choice sets are as follows:

```
      _
      A
    S l                   x x x x x x x x x x x x x x x x
    e t x x x x x x x x x 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2
    t _ 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5

    1 1 1 1 1 1 1 2 1 3 1 1 2 3 2 3 2 3 2 2 2 1 2 2 1 1 1
      2 2 2 1 1 3 2 1 3 3 1 3 2 2 3 2 1 2 3 3 1 1 2 3 2 3

    2 1 1 1 1 1 1 3 1 1 2 2 3 2 2 2 1 2 3 1 1 3 1 2 1 1 2
      2 3 3 1 1 1 1 3 3 2 2 2 1 2 2 2 3 1 1 3 3 1 2 1 2 2

  134 1 3 3 2 3 3 2 1 1 1 1 1 1 2 2 1 3 2 2 1 3 3 1 2 1 2
      2 1 1 2 2 1 2 1 1 1 1 1 3 2 2 3 1 1 2 1 1 3 3 1 3 3

  135 1 3 3 3 1 3 1 1 1 1 2 2 3 1 2 3 3 1 3 2 1 2 1 2 3 1
      2 2 1 1 1 1 1 1 3 1 2 2 1 3 2 1 3 3 1 2 1 2 1 2 2 2
```

Where the Restrictions Macro Gets Called

There is one more aspect to restrictions that must be understood for the most sophisticated usages of restrictions. The macro that imposes the restrictions is defined and called in four distinct places in the `%MktEx` macro. First, the restrictions macro is called in a separate, preliminary IML step, just to catch some syntax errors that you might have made. Next, it is called in between calling PROC PLAN or PROC FACTEX and calling PROC OPTEX. Here, the restrictions macro is used to impose restrictions on the candidate set. Next, it is used in the obvious way during design creation and the coordinate-exchange algorithm. Finally, when `options=accept` is specified, which means that restriction violations are acceptable, the macro is called after all of the iterations have completed to report on restriction violations in the final design. For some advanced restrictions, we might not want exactly the same code running in all four places. When the restrictions are purely written in terms of restrictions on `x`, which is the *ith* row of the design matrix, there is no problem. The same macro will work fine for all uses. However, when `xmat` (the full `x` matrix) or `i` or `j1` (the row or column number) are used, the same code typically cannot be used for all applications, although sometimes it does not matter. Next are some notes on each of the four phases.

Syntax Check. In this phase, the macro is defined and called just to check for syntax errors. This step lets the macro end more gracefully when there are errors and provides better information about the nature of the error than it would otherwise. Your restrictions macro can recognize when it is in this phase because the macro variable `&main` is set to 0 and the macro variable `&pass` is set to null. The pass variable is null before the iterations begin, 1 for the algorithm search phase, 2 for the design search phase, 3 for the design refinement stage, and 4 after the iterations end. You can conditionally execute code in this step or not using the following macro statements:

```
%if      &main eq 0 and &pass eq  %then %do;  /* execute in syntax check     */
%if not (&main eq 0 and &pass eq) %then %do;  /* not execute in syntax check */
```

You will usually not need to worry about this step. It just calls the macro once and ignores the results to check for syntax errors. For this step, `xmat` is a matrix of ones, and `x` is a vector of ones (since the design does not exist yet) and `j1 = j2 = j3 = i = 1`. If you have complicated restrictions involving the row or column exchange indices (`i`, `j1`, `j2`, `j3`) you might need to worry about this step. You might need to either not execute your restrictions in this step or *conditionally* execute some assignment statements (just for this step) that set up `j1`, `j2`, and `j3` more appropriately. Sometimes, you can set things up appropriately by using the `resmac=` option. Be aware however, that this step checks (`i`, `try`, `j1`, `j2`, `j3`, `x`, and `xmat` after your macro is called to ensure that you are not changing them because this is usually a sign of an error. If you get the following warning, make sure you are not incorrectly changing one of the matrices that you should not be changing:

```
WARNING: Restrictions macro is changing i, try, j1, j2, j3, x, or xmat.
         This might be a serious problem.  Check your macro.
```

If this step detects a syntax error, it will try to tell you where it is and what the problem is. If you have syntax errors in your restrictions macro and you cannot figure out what they are, sometimes the best thing to do is directly submit the statements in your restrictions macro to IML to so you can see the syntax errors. First, you need to submit the following statements:

```
%let n = 27; /* substitute number of runs    */
%let m = 10; /* substitute number of factors */
proc iml;
   xmat = j(&n, &m, 1);
   i = 1; j1 = 1; j2 = 1; j3 = 1; bad = 0; x = xmat[i,];
```

Candidate Check. In this phase, the macro is used to impose restrictions on the candidate set created by PROC PLAN or PROC FACTEX before it is searched by PROC OPTEX. The macro is called once for each row with the column index, `j1` set to 1. For some problems, such as most partial-profile problems, the restrictions are so severe that virtually none of the candidates will conform. Also, restrictions that are based on row number and column number do not make sense in the context of a candidate design. Your restrictions macro can recognize when it is in this phase because the macro variable `&main` is set to 0 and the macro variable `&pass` is set to 1 or 2. You can conditionally execute code in this step or not by using the following macro statements:

```
%if      &main eq 0 and &pass ge 1 and &pass le 2
         %then %do;                           /* execute on candidates     */
%if not (&main eq 0 and &pass ge 1 and &pass le 2)
         %then %do;                           /* not execute on candidates */
```

For simple restrictions not involving the column exchange indices (j1, j2, j3), you probably do not need to worry about this step. If you use j1, j2, or j3, you will need to either not execute your restrictions in this step or conditionally execute some assignment statements that set up j1, j2, and j3 appropriately. Ordinarily for this step, xmat contains the candidate design, x contains the *ith* row, j1 = 0; j2 = 0; j3 = 0; try = 1; i is set to the candidate row number.

Main Coordinate-Exchange Algorithm. In this phase, the macro is used to impose restrictions on the design as it is being built in the coordinate-exchange algorithm. Your restrictions macro can recognize when it is in this phase because the macro variable &main is set to 1 and the macro variable &pass is set to 1, 2, or 3. You can conditionally execute code in this step or not by using the following macro statements:

```
%if      &main eq 1 and &pass ge 1 and &pass le 3
         %then %do;                   /* execute on coordinate exchange     */
%if not (&main eq 1 and &pass ge 1 and &pass le 3)
         %then %do;                   /* not execute on coordinate exchange */
```

For this step, xmat contains the candidate design, x contains the *ith* row, j1, j2, and j3 typically contain the column indices, i is the row number, and try is the zero-based cumulative row number. With exchange=1, j1 exists, with exchange=2, j1 and j2 exist, and so on. Sometimes in this phase, the restrictions macro is called once per row with the j* indices all set to 1. If you use the j* indices in your restrictions, you might need to allow for this. For example, if you are checking the current j1 column for balance, and you used an init= data set with column one fixed and unbalanced, you will not want to perform the check when j1 = 1. Note that for some designs that are partially initialized with an orthogonal array and for some uses of init=, not all columns or cells in the design are evaluated.

Restrictions Violations Check. In this phase, the macro is used to check the design when there are restrictions and options=accept. The macro is called once for each row of the design. Your restrictions macro can recognize when it is in this phase because the macro variable &main is set to 1 and the macro variable &pass is greater than 3. You can conditionally execute code in this step or not by using the following macro statements:

```
%if      &main eq 1 and &pass gt 3  %then %do; /* execute on final check     */
%if not (&main eq 1 and &pass gt 3) %then %do; /* not execute on final check */
```

For this step, xmat contains the candidate design, x contains the *ith* row, j1 = 1; j2 = 1; j3 = 1; try = 1; and i is the row number.

Using the following macro is equivalent to specifying `partial=4`:

```
%macro partprof;
   nvary = sum(x ^= 1);
   %if &main %then %do;
      if i = 1 then bad = nvary;
      else          bad = abs(nvary - 4);
      %end;
   %else %do;
      bad = ^ (nvary = 0 | nvary = 4);
      %end;
   %mend;
```

In the main algorithm, when imposing restrictions on the design, we restrict the first run to be constant and all other runs to have four attributes varying. For the candidate-set restrictions, when MAIN is zero, any observation with zero or four varying factors is acceptable. For the candidate-set restrictions, there is no reason to count the number of violations. A candidate run is either acceptable or not. We do not worry about the syntax error or final check steps; both versions will work fine in either.

%MktKey Macro

The `%MktKey` autocall macro creates expanded lists of variable names. See the following pages for examples of using this macro in the design chapter: 133 and 192. Also see the following pages for examples of using this macro in the discrete choice chapter: 356, 546, 556, 575, 607, 617, 628, and 636. Additional examples appear throughout this chapter. You can specify the number of rows followed by a number of columns. The output is a data set called KEY. This is illustrated in the following step:

```
   %mktkey(5 10)
```

The KEY output data set with 5 rows and 10 columns and $5 \times 10 = 50$ variable names, `x1-x50` is as follows:

```
          x1     x2     x3     x4     x5     x6     x7     x8     x9     x10

          x1     x2     x3     x4     x5     x6     x7     x8     x9     x10
          x11    x12    x13    x14    x15    x16    x17    x18    x19    x20
          x21    x22    x23    x24    x25    x26    x27    x28    x29    x30
          x31    x32    x33    x34    x35    x36    x37    x38    x39    x40
          x41    x42    x43    x44    x45    x46    x47    x48    x49    x50
```

Alternatively, you can specify the number of rows and number of columns followed by a `t` or `T` and get the transpose of this data set. The output data set is again called KEY. The following step illustrates this option:

```
   %mktkey(5 10 t)
```

The KEY output data set with 5 rows and 10 columns and $5 \times 10 = 50$ variable names, `x1-x50` is as follows:

```
          x1    x2     x3     x4     x5     x6     x7     x8     x9     x10

          x1    x6     x11    x16    x21    x26    x31    x36    x41    x46
          x2    x7     x12    x17    x22    x27    x32    x37    x42    x47
          x3    x8     x13    x18    x23    x28    x33    x38    x43    x48
          x4    x9     x14    x19    x24    x29    x34    x39    x44    x49
          x5    x10    x15    x20    x25    x30    x35    x40    x45    x50
```

Note that this time the names progress down the columns instead of across the rows.

The `%MktKey` macro has another type of syntax as well. You can provide the `%MktKey` macro with a list of variables as follows:

```
   %mktkey(x1-x15)
```

The `%MktKey` macro produced the following line:

```
x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15
```

You can copy and paste this list to make it easier to construct the `key=` data set for the `%MktRoll` macro. The following step makes the `Key` data set:

```
data key;
   input (x1-x5) ($);
   datalines;
 x1  x2  x3  x4  x5
 x6  x7  x8  x9 x10
x11 x12 x13 x14 x15
  .   .   .   .   .
;
```

Alternatively, if you want to use precisely the `Key` data set that the `%MktKey` macro creates, you can have the `%MktRoll` macro automatically construct the `key=` data set for you by specifying the same argument in the `key=` option that you would specify in the `%MktKey` macro. In the sample code below, the first two steps are equivalent to the third step:

```
%mktkey(3 3)
%mktroll(design=design, key=key, out=rolled)

%mktroll(design=design, key=3 3, out=rolled)
```

%MktKey Macro Options

The following option can be used with the `%MktKey` macro:

MktKey Macro Options

Option	Description
`list`	(positional) variable list or n rows and n columns
	(positional) “help” or “?” displays syntax summary

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%mktkey(help)
%mktkey(?)
```

The only argument to the `%MktKey` macro is the list.

list

specifies the variable list or matrix size list. Note that the list is a positional parameter, hence it is not specified after a name and an equal sign. The list can be a variable list.

Alternatively, the list contains the number of rows followed by the number of columns, optionally followed by a `t` or `T` (for transpose). Without the `t` the names go `x1`, `x2`, `x3`, ..., across each row. With the `t` the names go `x1`, `x2`, `x3`, ..., down each column.

%MktLab Macro

The %MktLab autocall macro processes an experimental design, usually created by the %MktEx macro, and assigns the final variable names and levels. See the following pages for examples of using this macro in the design chapter: 81, 85, 109, 112, 166 and 201. Also see the following pages for examples of using this macro in the discrete choice chapter: 333, 353, 501, 538, 564, 567, 596 and 602. Additional examples appear throughout this chapter. For example, say you used the %MktEx macro to create a design with 11 two-level factors (with default levels of 1 and 2). The following steps create and display the design:

```
%mktex(n=12, options=nosort)

proc print noobs; run;
```

The design is as follows:

```
x1   x2   x3   x4   x5   x6   x7   x8   x9   x10   x11

 1    1    1    1    1    1    1    1    1    1     1
 1    2    2    2    1    1    2    1    1    2     2
 2    2    1    1    2    1    2    2    1    2     1
 1    1    1    2    1    2    2    2    2    2     1
 1    2    2    2    2    1    1    2    2    1     1
 1    2    1    1    2    2    2    1    2    1     2
 2    2    1    2    1    2    1    2    1    1     2
 2    1    1    2    2    1    1    1    2    2     2
 2    1    2    2    2    2    2    1    1    1     1
 1    1    2    1    2    2    1    2    1    2     2
 2    2    2    1    1    2    1    1    2    2     1
 2    1    2    1    1    1    2    2    2    1     2
```

Either the %MktEx macro or the %MktLab macro can be used to assign levels of –1 and 1 and add an intercept. You can do it directly with the %MktEx macro, using `levels=i int`. The value of `i` specifies centered integer levels, and `int` adds the intercept. The following step illustrates this:

```
%mktex(n=12,                       /* 12 runs                            */
       options=nosort,             /* do not sort design                 */
       levels=i                    /* -1 and 1 instead of 1 and 2        */
              int)                 /* add an intercept to the design     */
```

However, if you want to change the factor names, and for more complicated relabeling of the levels, you need to use the %MktLab macro. This is illustrated in the following step:

```
%mktex(n=12, options=nosort)

%mktlab(data=design, values=1 -1, int=Had0, prefix=Had)

proc print noobs; run;
```

The `%MktLab` macro assigns levels of –1 and 1, adds an intercept named `Had0`, and changes the variable name prefixes from `x` to `Had`. This creates a Hadamard matrix (although, of course, the Hadamard matrix can have any set of variable names). The resulting Hadamard matrix is as follows:

```
      Had0  Had1  Had2  Had3  Had4  Had5  Had6  Had7  Had8  Had9  Had10  Had11

        1    -1    -1    -1    -1    -1    -1    -1    -1    -1     -1     -1
        1     1    -1     1    -1    -1    -1     1     1     1     -1      1
        1     1     1    -1     1    -1    -1    -1     1     1      1     -1
        1    -1     1     1    -1     1    -1    -1    -1     1      1      1
        1     1    -1     1     1    -1     1    -1    -1    -1      1      1
        1     1     1    -1     1     1    -1     1    -1    -1     -1      1
        1     1     1     1    -1     1     1    -1     1    -1     -1     -1
        1    -1     1     1     1    -1     1     1    -1     1     -1     -1
        1    -1    -1     1     1     1    -1     1     1    -1      1     -1
        1    -1    -1    -1     1     1     1    -1     1     1     -1      1
        1     1    -1    -1    -1     1     1     1    -1     1      1     -1
        1    -1     1    -1    -1    -1     1     1     1    -1      1      1
```

Alternatively, you can use the `key=` data set that follows to do the same thing:

```
data key;
  array Had[11];
  input Had1 @@;
  do i = 2 to 11; Had[i] = Had1; end;
  drop i;
  datalines;
1 -1
;

proc print data=key; run;
```

The `key=` data set is as follows:

```
      Obs  Had1  Had2  Had3  Had4  Had5  Had6  Had7  Had8  Had9  Had10  Had11

        1     1     1     1     1     1     1     1     1     1      1      1
        2    -1    -1    -1    -1    -1    -1    -1    -1    -1     -1     -1
```

The following step uses this data set to make the final design:

```
%mktlab(data=design, key=key, int=Had0)
```

The Hadamard matrix from this step (not shown) is exactly the same as the one just shown previously.

The `key=` data set contains all of the variables that you want in the design and all of their levels. This information is applied to the design, by default the one stored in a data set called RANDOMIZED, which is the default `outr=` data set name from the `%MktEx` macro. The results are stored in a new data set, FINAL, with the desired factor names and levels.

Consider the consumer food product example from page 255. The following step reads a possible design:

```
data randomized;
  input x1-x8 @@;
  datalines;
4 2 1 1 1 2 2 2 2 1 1 2 1 3 1 3 3 4 2 2 1 3 2 3 4 3 2 1 3 2 2 3 4 1 2 1
1 1 1 1 2 4 1 2 1 2 1 1 1 2 1 2 3 3 2 1 2 2 2 2 2 2 2 3 1 4 2 1 1 2 2 2
3 2 2 1 3 1 2 1 1 4 1 2 2 3 1 2 1 3 2 2 1 3 1 1 3 2 1 2 2 1 2 3 3 4 1 1
3 1 1 3 4 1 2 2 2 1 2 1 2 3 2 1 2 3 2 2 2 1 2 1 3 3 1 3 4 2 2 2 1 3 1 2
2 4 2 2 3 1 1 2 3 1 2 2 3 2 1 2 3 3 1 1 2 3 1 1 4 4 2 1 2 2 1 3 1 1 1 1
3 2 1 2 4 3 1 2 3 3 2 2 1 2 2 1 2 1 1 3 1 3 1 1 1 1 2 3
;
```

Designs created by the `%MktEx` macro always have factor names `x1`, `x2`, ..., and so on, and the levels are consecutive integers beginning with 1 (1, 2 for two-level factors; 1, 2, 3 for three-level factors; and so on). The `%MktLab` macro provides you with a convenient way to change the names and levels to more meaningful values. The data set KEY contains the variable names and levels that you ultimately want. The following step creates a `Key` data set:

```
data key;
  missing N;
  input Client ClientLineExtension ClientMicro $ ShelfTalker $
        Regional Private PrivateMicro $ NationalLabel;
  format _numeric_ dollar5.2;
  datalines;
1.29 1.39 micro Yes 1.99 1.49 micro 1.99
1.69 1.89 stove No  2.49 2.29 stove 2.39
2.09 2.39 .     .   N    N    .     N
N    N    .     .   .    .    .     .
;

%mktlab(data=randomized, key=key)

proc sort; by shelftalker; run;

proc print; by shelftalker; run;
```

The variable `Client` with 4 levels is made from `x1`, `ClientLineExtension` with 4 levels is made from `x2`, `ClientMicro` with 2 levels is made from `x3`. The `N` (for not available) is treated as a special missing value. The `Key` data set has four rows because the maximum number of levels is four. Factors with fewer than four levels are filled in with ordinary missing values. The `%MktLab` macro takes the default `data=randomized` data set from `%MktEx` and uses the rules in the `key=key` data set, to create the information in the `out=final` data set, which is shown next, sorted by the shelf talker variable.

Some of the design is as follows:

```
----------------------------- ShelfTalker=No -----------------------------

                 Client
                  Line      Client                           Private   National
Obs    Client   Extension   Micro     Regional   Private     Micro     Label

  1    $1.69      $1.39     micro      $1.99        N        micro       N
  2    $2.09        N       stove      $1.99        N        stove       N
  3    $1.69        N       micro      $1.99      $2.29      micro     $1.99
  .
  .
  .

----------------------------- ShelfTalker=Yes ----------------------------

                 Client
                  Line      Client                           Private   National
Obs    Client   Extension   Micro     Regional   Private     Micro     Label

 14      N        $1.89     micro      $1.99      $2.29      stove     $2.39
 15      N        $2.39     stove        N        $2.29      stove       N
 16      N        $1.39     stove      $1.99      $1.49      micro     $1.99
  .
  .
  .
```

This macro creates the `out=` data set by repeatedly reading and rereading the `key=` data set, one datum at a time, using the information in the `data=` data set to determine which levels to read from the `key=` data set. In this example, for the first observation, `x1`=4 so the fourth value of the first `key=` variable is read, then `x2`=2 so the second value of the second `key=` variable is read, then `x3`=1 so the first value of the third `key=` variable is read, ..., then `x8`=2 so the second value of the eighth `key=` variable is read, then the first observation is output. This continues for all observations. Unlike previous releases, the `data=` data is not required to have the default `levels=` specification (integer values beginning with 1). Other numeric values are fine and are converted to consecutive positive integers before performing the final mapping.

The following steps create the L_{36}, changes the names of the two-level factors to `two1-two11`, assigns them values the –1, 1, changes the names of the three-level factors to `thr1-thr12`, and assigns them the values –1, 0, 1:

```
%mktex(n=36, seed=420)

data key;
  array x[23] two1-two11 thr1-thr12;
  input two1 thr1;
  do i =  2 to 11; x[i] = two1; end;
  do i = 13 to 23; x[i] = thr1; end;
  drop i;
  datalines;
-1 -1
 1  0
 .  1
;

%mktlab(data=randomized, key=key)

proc print data=key noobs; var two:; run;
proc print data=key noobs; var thr:; run;

proc print data=final(obs=5) noobs; var two:; run;
proc print data=final(obs=5) noobs; var thr:; run;
```

The `Key` data set is as follows:

```
  two1   two2   two3   two4   two5   two6   two7   two8   two9   two10   two11

   -1     -1     -1     -1     -1     -1     -1     -1     -1     -1      -1
    1      1      1      1      1      1      1      1      1      1       1
    .      .      .      .      .      .      .      .      .      .       .

    thr1  thr2  thr3  thr4  thr5  thr6  thr7  thr8  thr9  thr10  thr11  thr12

     -1    -1    -1    -1    -1    -1    -1    -1    -1     -1     -1     -1
      0     0     0     0     0     0     0     0     0      0      0      0
      1     1     1     1     1     1     1     1     1      1      1      1
```

The first five rows of the design are as follows:

```
  two1   two2   two3   two4   two5   two6   two7   two8   two9   two10   two11

   -1     -1      1      1      1      1      1      1      1      1      -1
    1      1      1      1     -1      1      1     -1     -1      1       1
   -1     -1     -1      1     -1     -1     -1     -1     -1      1      -1
   -1     -1     -1     -1      1      1      1     -1     -1     -1       1
   -1      1     -1     -1     -1     -1      1      1      1      1       1
   thr1  thr2  thr3  thr4  thr5  thr6  thr7  thr8  thr9  thr10  thr11  thr12

     0     0     0     1     1     1    -1     1     1      1     -1     -1
     1    -1    -1     1     0     0    -1     1     1      0      1      0
     0    -1     0     0    -1    -1    -1     0    -1      0     -1      0
     0     1     0     1     1     0     1    -1    -1     -1      1      0
     1    -1     1     1     1     1     0    -1    -1      0     -1      1
```

This following steps create a design and block it. This example shows that it is okay if not all of the variables in the input design are used. The variables `Block`, `Run`, and `x4` are just copied from the input to the output. The following steps create the design:

```
%mktex(n=18, seed=396)

%mktblock(data=design, nblocks=2, factors=x1-x4, seed=292)

data key;
   input Brand $ Price Size;
   format price dollar5.2;
   datalines;
Acme 1.49   6
Apex 1.79   8
.    1.99  12
;

%mktlab(data=blocked, key=key)

proc print; run;
```

The results are as follows:

```
                     Block    Run    Brand    Price    Size    x4

                       1       1     Acme     $1.49     12      2
                               2     Acme     $1.79      6      3
                               3     Acme     $1.79      8      2
                               4     Acme     $1.99      8      1
                               5     Acme     $1.99     12      1
                               6     Apex     $1.49      8      3
                               7     Apex     $1.49     12      3
                               8     Apex     $1.79      6      1
                               9     Apex     $1.99      6      2

                       2       1     Acme     $1.49      6      1
                               2     Acme     $1.49      6      2
                               3     Acme     $1.79      8      3
                               4     Acme     $1.99     12      3
```

```
                              5     Apex     $1.49      8       1
                              6     Apex     $1.79     12       1
                              7     Apex     $1.79     12       2
                              8     Apex     $1.99      6       3
                              9     Apex     $1.99      8       2
```

This next example illustrates using the `labels=` option. This option is more typically used with `values=` input, rather than when you construct the `key=` data set yourself, but it can be used either way. This example is from a vacation choice example. The following steps create the design:

```
%mktex(3 ** 15,                       /* 15 three-level factors               */
       n=36,                          /* 36 runs                              */
       seed=17,                       /* random number seed                   */
       maxtime=0)                     /* no more than 1 iter in each phase    */

%mktblock(data=randomized, nblocks=2, factors=x1-x15, seed=448)

%macro lab;
   label X1  = 'Hawaii, Accommodations'
         X2  = 'Alaska, Accommodations'
         X3  = 'Mexico, Accommodations'
         X4  = 'California, Accommodations'
         X5  = 'Maine, Accommodations'
         X6  = 'Hawaii, Scenery'
         X7  = 'Alaska, Scenery'
         X8  = 'Mexico, Scenery'
         X9  = 'California, Scenery'
         X10 = 'Maine, Scenery'
         X11 = 'Hawaii, Price'
         X12 = 'Alaska, Price'
         X13 = 'Mexico, Price'
         X14 = 'California, Price'
         X15 = 'Maine, Price';
   format x11-x15 dollar5.;
%mend;
```

```
data key;
   length x1-x5 $ 16 x6-x10 $ 8 x11-x15 8;
   input x1 & $ x6 $ x11;
   x2  = x1;    x3 = x1;    x4 = x1;    x5 = x1;
   x7  = x6;    x8 = x6;    x9 = x6;   x10 = x6;
   x12 = x11;  x13 = x11;  x14 = x11;  x15 = x11;
   datalines;
Cabin            Mountains   999
Bed & Breakfast  Lake       1249
Hotel            Beach      1499
;

%mktlab(data=blocked, key=key, labels=lab)

proc contents p; ods select position; run;
```

The variable name, label, and format information are as follows:

```
                                The CONTENTS Procedure

                              Variables in Creation Order

        #    Variable    Type    Len    Format      Label

        1    x1          Char     16                Hawaii, Accommodations
        2    x2          Char     16                Alaska, Accommodations
        3    x3          Char     16                Mexico, Accommodations
        4    x4          Char     16                California, Accommodations
        5    x5          Char     16                Maine, Accommodations
        6    x6          Char      8                Hawaii, Scenery
        7    x7          Char      8                Alaska, Scenery
        8    x8          Char      8                Mexico, Scenery
        9    x9          Char      8                California, Scenery
       10    x10         Char      8                Maine, Scenery
       11    x11         Num       8    DOLLAR5.    Hawaii, Price
       12    x12         Num       8    DOLLAR5.    Alaska, Price
       13    x13         Num       8    DOLLAR5.    Mexico, Price
       14    x14         Num       8    DOLLAR5.    California, Price
       15    x15         Num       8    DOLLAR5.    Maine, Price
       16    Block       Num       8
       17    Run         Num       8
```

%MktLab Macro Options

The following options can be used with the `%MktLab` macro:

Option	Description
`help`	(positional) "help" or "?" displays syntax summary
`cfill=`*character-string*	character fill value
`data=`*SAS-data-set*	input design data set
`dolist=`*do-list*	new values using a do-list syntax
`int=`*variable-list*	name of an intercept variable
`key=`*SAS-data-set*	`Key` data set
`labels=`*macro-name*	macro that provides labels and formats
`nfill=`*number*	numeric fill value
`options=noprint`	suppress the display of the variable mappings
`out=`*SAS-data-set*	output data set with recoded design
`prefix=`*variable-prefix*	prefix for naming variables
`statements=`*SAS-code*	add extra statements
`values=`*value-list*	the new values for all of the variables
`vars=`*variable-list*	list of variable names

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%mktlab(help)
%mktlab(?)
```

cfill= *character-string*

specifies the fill value in the `key=` data set for character variables. See the `nfill=` option for more information about fill values. The default is `cfill=' '`.

data= *SAS-data-set*

specifies the input data set with the experimental design, usually created by the `%MktEx` macro. The default is `data=Randomized`. The factor levels in the `data=` data set must be consecutive integers beginning with 1.

dolist= *do-list*

specifies the new values, using a do-list syntax (`n TO m <BY p>`), for example: `dolist=1 to 10` or `dolist=0 to 9`. With asymmetric designs (not all factors have the same levels), specify the levels for the largest number of levels. For example, with two-level and three-level factors and `dolist= 0 to 2`, the two-level factors is assigned levels 0 and 1, and the three-level factors is assigned levels 0, 1, and 2. Do not specify both `values=` and `dolist=`. By default, when `key=`, `values=`, and `dolist=` are all not specified, the default value list comes from `dolist=1 to 100`.

int= *variable-list*

specifies the name of an intercept variable (column of ones), if you want an intercept added to the `out=`

data set. You can also specify a variable list instead of a variable name if you would like to make a list of variables with values all one. This can be useful, for example, for creating flag variables for generic choice models when the design is going to be used as a candidate set for the `%ChoicEff` macro.

key= *SAS-data-set*

specifies the input data set with the key to recoding the design. When `values=` or `dolist=` is specified, this data set is made for you. By default, when `key=`, `values=`, and `dolist=` are all not specified, the default value list comes from `dolist=1 to 100`.

labels= *macro-name*

specifies the name of a macro that provides labels, formats, or other additional information to the `key=` data set. For a simple format specification, it is easier to use `statements=`. For more involved specifications, use `labels=`. Note that you specify just the macro name, no percents on the `labels=` option. This option is illustrated in the following step:

```
%mktex(3 ** 4, n=18, seed=205)

%macro labs;
   label x1 = 'Sploosh' x2 = 'Plumbob'
         x3 = 'Platter' x4 = 'Moosey';
   format x1-x4 dollar5.2;
   %mend;

%mktlab(data=randomized, values=1.49 1.99 2.49, labels=labs)

proc print label; run;
```

The first part of the design is as follows:

```
              Obs    Sploosh    Plumbob    Platter    Moosey

                1     $2.49      $2.49      $2.49     $1.49
                2     $2.49      $2.49      $1.99     $1.99
                3     $1.49      $1.49      $1.49     $1.49
                4     $1.99      $1.99      $2.49     $2.49
                .
                .
                .
```

nfill= *number*

specifies the fill value in the `key=` data set for numeric variables. For example, when the maximum number of levels is three, the last value in the `key=` data set for numeric two-level factors should have a value of `nfill=`, which by default is ordinary missing. If the macro tries to access one of these values, it displays a warning. If you would like ordinary missing (.) to be a legitimate level, specify a different `nfill=` value and use it for the extra places in the `key=` data set.

options= *options-list*

specifies binary options. By default, none of these options are specified. Specify values after `options=`.

> `noprint`
> suppresses the display of the variable mappings.

out= *SAS-data-set*

specifies the output data set with the final, recoded design. The default is `out=final`. Often, you will want to specify a two-level name to create a permanent SAS data set so the design is available later for analysis.

prefix= *variable-prefix*

specifies a prefix for naming variables when `values=` is specified. For example, `prefix=Var` creates variables `Var1`, `Var2`, and so on. By default, the variables are `x1`, `x2`, ... This option is ignored when `vars=` is specified.

statements= *SAS-code*

is an alternative to `labels=` that you can use to add extra statements to the `key=` data set. For a simple format specification, it is easier to use `statements=`. For more involved specifications, use `labels=`. This option is illustrated in the following step:

```
%mktex(3 ** 4, n=18, seed=205)

%mktlab(data=randomized, values=1.49 1.99 2.49,
        vars=Sploosh Plumbob Platter Moosey,
        statements=format Sploosh Plumbob Platter Moosey dollar5.2)

proc print; run;
```

The first part of the design is as follows:

```
Obs    Sploosh    Plumbob    Platter    Moosey

  1     $2.49      $2.49      $2.49     $1.49
  2     $2.49      $2.49      $1.99     $1.99
  3     $1.49      $1.49      $1.49     $1.49
  4     $1.99      $1.99      $2.49     $2.49
  .
  .
  .
```

values= *value-list*

specifies the new values for all of the variables. If all variables will have the same value, it is easier to specify `values=` or `dolist=` than `key=`. When you specify `values=`, the `key=` data set is created for you. Specify a list of levels separated by blanks. If your levels contain blanks, separate them with two blanks. With asymmetric designs (not all factors have the same levels) specify the levels for the largest number of levels. For example, with two-level and three-level factors and `values=a b c`, the two-level factors are assigned levels `'a'` and `'b'`, and the three-level factors are assigned levels `'a'`, `'b'`, and `'c'`. Do not specify both `values=` and `dolist=`. By default, when `key=`, `values=`, and `dolist=` are all not specified, the default value list comes from `dolist=1 to 100`.

vars= *variable-list*
specifies a list of variable names when `values=` or `dolist=` is specified. If `vars=` is not specified with `values=`, then `prefix=` is used.

%MktLab Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktMDiff Macro

The `%MktMDiff` autocall macro analyzes MaxDiff (maximum difference or best-worst) data (Louviere 1991, Finn and Louviere 1992). The result of the analysis is a scaling of the attributes on a preference or importance scale. In a MaxDiff study, subjects are shown sets of messages or product attributes and are asked to choose the best (or most important) from each set as well as the worst (or least important). The design consists of:

- *t* attributes
- *b* sets (or blocks) of attributes with
- *k* attributes in each set

These are the parameters of a balanced incomplete block design or BIBD, which can be constructed from the `%MktBIBD` macro. Note, however, that you can use designs that are produced by the `%MktBIBD` macro but do not meet the strict requirements for a BIBD. See page 1111 for more information about this. To illustrate, a researcher is interested in preference for cell phones based on the attributes of the phones. The attributes are as follows:

Camera
Flip
Hands Free
Games
Internet
Free Replacement
Battery Life
Large Letters
Applications

Subjects are shown subsets of these 9 attributes and asked to pick which is the most important when they make a cell phone choice and which is the least important. We can use the `%MktBSize` macro as follows to get ideas about how many blocks to use and how many to show at one time:

```
%mktbsize(nattrs=9, setsize=2 to 9, nsets=1 to 20)
```

The results of this step are as follows:

```
                                                                      n
t               k          b            r              Lambda        Total
Number of       Set        Number       Attribute      Pairwise      Sample
Attributes      Size       of Sets      Frequency      Frequencies   Size

    9            3            12            4              1            36
    9            4            18            8              3            72
    9            5            18           10              5            90
    9            6            12            8              5            72
    9            8             9            8              7            72
```

With 9 attributes, there are five sizes that meet the necessary but not sufficient conditions for the existence of a BIBD. Of the candidates (3, 4, 5, 6, and 8), 4 or 5 seem like good choices. (Three seems a bit small and more than 5 seems a bit big given that we only have 9 attributes.) The following step creates a BIBD with $t = 9$ attributes, shown in $b = 18$ sets of size $k = 5$:

```
%mktbibd(nattrs=9, setsize=5, nsets=18, seed=377, out=sasuser.bibd)
```

The following table, produced by the macro, shows that the design is in fact a BIBD:

```
                    Attribute by Attribute Frequencies

                          1  2  3  4  5  6  7  8  9

                       1 10  5  5  5  5  5  5  5  5
                       2    10  5  5  5  5  5  5  5
                       3       10  5  5  5  5  5  5
                       4          10  5  5  5  5  5
                       5             10  5  5  5  5
                       6                10  5  5  5
                       7                   10  5  5
                       8                      10  5
                       9                         10
```

This is the attribute by attribute frequency matrix. The diagonal elements of the matrix show how often each attribute occurs. Each of the $t = 9$ attributes occurs the same number of times (10 times). Furthermore, each of the $t = 9$ attributes occurs with each of the remaining 8 attributes exactly 5 times. These two constant values, one one the diagonal and one off, show that the design is a BIBD. The design is as follows:

```
                    Balanced Incomplete Block Design

                       x1    x2    x3    x4    x5

                        1     2     4     7     9
                        6     2     9     8     4
                        5     8     2     1     7
                        7     9     1     6     3
                        9     7     6     5     2
                        3     1     5     6     8
                        4     6     3     8     2
                        6     7     4     1     8
                        7     5     2     3     4
                        1     3     8     2     7
                        3     4     1     5     9
                        2     9     6     3     1
                        8     3     9     2     5
                        9     8     7     4     3
                        5     4     8     9     1
```

```
2   1   5   4   6
8   5   7   9   6
4   6   3   7   5
```

The first set or consists of attributes 1, 2, 4, 7, and 9, which are as follows:

Camera
Flip
Games
Battery Life
Applications

Subjects will choose the best and worst from this and every other set.

The `%MktMDiff` macro reads the experimental design and the data, combines them, arranges them in the right form for analysis, and performs the analysis using a multinomial logit model. The data are arrayed so that each original MaxDiff set forms two choice sets in the analysis: one positively weighted set for the best choice and one negatively weighted set for the worst choice. The input data can come in one of eight forms:

`bw`
best then worst (e.g. `b1-b18 w1-w18`), and the data are attribute numbers (range from 1 to `nattrs=`t). If the `groups=` option is specified, for example with `groups=3`, the variables are `b1-b6 w1-w6` and three observations provide the information about all of the rows in the block design.

`wb`
worst then best (e.g. `w1-w18 b1-b18`), and the data are attribute numbers (range from 1 to `nattrs=`t). If the `groups=` option is specified, for example with `groups=3`, the variables are `w1-w6 b1-b6` and three observations provide the information about all of the rows in the block design.

`bwalt`
best then worst and alternating (e.g. `b1 w1 b2 w2 ... b18 w18`), and the data are attribute numbers (range from 1 to `nattrs=`t). If the `groups=` option is specified, for example with `groups=3`, the variables are `b1 w1 b2 w2 ... b6 w6` and three observations provide the information about all of the rows in the block design.

`wbalt`
worst then best and alternating (e.g. `w1 b1 w2 b2 ... w18 b18`), and the data are attribute numbers (range from 1 to `nattrs=`t). If the `groups=` option is specified, for example with `groups=3`, the variables are `w1 b1 w2 b2 ... w6 b6` and three observations provide the information about all of the rows in the block design.

`bwpos`
best then worst (e.g. `b1-b18 w1-w18`), and the data are positions (range from 1 to `setsize=`k). If the `groups=` option is specified, for example with `groups=3`, the variables are `b1-b6 w1-w6` and three observations provide the information about all of the rows in the block design.

`wbpos`
worst then best (e.g. `w1-w18 b1-b18`), and the data are positions (range from 1 to `setsize=`k). If the `groups=` option is specified, for example with `groups=3`, the variables are `w1-w6 b1-b6` and three observations provide the information about all of the rows in the block design.

`bwaltpos`
best then worst and alternating (e.g. `b1 w1 b2 w2 ... b18 w18`), and the data are positions (range from 1 to `setsize=`k). If the `groups=` option is specified, for example with `groups=3`, the variables are `b1 w1 b2 w2 ... b6 w6` and three observations provide the information about all of the rows in the block design.

`wbaltpos`
worst then best and alternating (e.g. `w1 b1 w2 b2 ... w18 b18`), and the data are positions (range from 1 to `setsize=`k). If the `groups=` option is specified, for example with `groups=3`, the variables are `w1 b1 w2 b2 ... w6 b6` and three observations provide the information about all of the rows in the block design.

Note that in all cases, any variable names can be used. For example, the variables could be `x1-x36`. In the case of `bwalt`, the odd numbered variables will correspond to best picks and the even numbered variables will correspond to worst picks.

The following example uses the design created previously:

```
title 'Best Worst Example with Cell Phone Attributes';

data bestworst;
   input Sub $ @4 (b1-b18 w1-w18) (1.);
   datalines;
 1 188661884399349653941955342212935494
 2 765358873891388493922673644336595554
 3 782126282892848564995993447213935655
 4 481363264246399187162125415351281453
 5 787168863811878175225995382352235293
 6 787658867891878667495965442313345453
 7 788171867736888187465395344393344453
 8 788771867711888687445353445353335254
 9 188778887896878687425323443242344454
10 788778887816888687445353444343344454
11 787778877816878667442353343343235453
12 787778387711898667425321443253544594
13 767668285791988687441951364232234194
14 187168877741878687445323445216334453
15 788176487116988675492393314339579457
16 267665615733884677442193342349545564
17 481191867813938266147956244139584594
18 725778814832585185141193643296944467
19 188678863811279263445123344253934596
20 728698612719281483265755285851944597
;
```

```
%let attrlist=Camera,Flip,Hands Free,Games,Internet
,Free Replacement,Battery Life,Large Letters,Applications;

%phchoice( on )

%mktmdiff(bw, nattrs=9, nsets=18, setsize=5, attrs=attrlist,
          data=bestworst, design=sasuser.bibd)
```

The DATA step reads 36 variables with data and a subject variable, which is ignored. The descriptions of each of the t attributes are listed in comma-delimited form and stored in a macro variable. (Note that when the list is split across lines, care is taken to ensure that the next attribute description, “Free Replacement” immediately follows the comma so that it will not begin with a leading blank.) The `%PHChoice` macro is used to customize the output from PROC PHREG, which the `%MktMDiff` macro calls, to look like the output from a discrete choice procedure instead of a survival analysis procedure. The `%MktMDiff` macro begins with a positional parameter that specifies the layout of the data. Positional parameters do not begin with a keyword and an equal sign. Because of the initial `bw` specification, the data are best then worst. The variables do not alternate. The data are attribute numbers not positions. The design has `nsets=18` sets, `nattrs=9` attributes, and `setsize=5` are shown at a time. The `attrlist` macro variable contains the list of the attributes. Note that a variable name and not the value of the variable are specified. In other words, `attrlist` not `&attrlist` is specified. The SAS data set with the data is called `bestworst`, and the SAS data set with the design is called `sasuser.bibd`.

The `%MktMDiff` macro begins by displaying the following summary of the input:

```
Var Order:   Best then Worst
Alternating: Variables Do Not Alternate
Data:        Attribute Numbers (Not Positions)
Best Vars:   b1 b2 b3 b4 b5 b6 b7 b8 b9 b10 b11 b12 b13 b14 b15 b16 b17 b18
Worst Vars:  w1 w2 w3 w4 w5 w6 w7 w8 w9 w10 w11 w12 w13 w14 w15 w16 w17 w18
Attributes:  Camera
             Flip
             Hands Free
             Games
             Internet
             Free Replacement
             Battery Life
             Large Letters
             Applications
```

The data consists of the best variables then the worst variables. The best variables are `b1-b18,` and the worst variables are `w1-w18`. The data are attribute numbers (1 to $t = 9$) not positions (1 to $k = 5$). Finally, the descriptions of each of the t attributes are listed.

The following table is of interest as a check of the integrity of the input data:

```
              Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

                         Number of       Number of          Chosen          Not
             Pattern       Choices    Alternatives    Alternatives       Chosen

                   1            36             100              20           80
```

In the aggregate data set, there is one pattern of input and it occurs 36 times (18 best choices per subject plus 18 worst choices). There are 100 alternatives (5 attributes in a set examined by 20 individuals), each of the 20 subjects chose 1 ($20 \times 1 = 20$) and did not choose 4 ($20 \times 4 = 80$).

The final table of results is as follows:

```
                          Best Worst Example with Cell Phone Attributes
                             Multinomial Logit Parameter Estimates

                                   Parameter      Standard                     Pr >
                           DF       Estimate         Error    Chi-Square      ChiSq

      Large Letters         0              0             .             .          .
      Battery Life          1       -0.52009       0.17088        9.2635     0.0023
      Free Replacement      1       -1.01229       0.17686       32.7598     <.0001
      Camera                1       -1.30851       0.18232       51.5096     <.0001
      Applications          1       -1.93107       0.18741      106.1680     <.0001
      Flip                  1       -2.00892       0.18788      114.3334     <.0001
      Internet              1       -2.44592       0.18731      170.5156     <.0001
      Hands Free            1       -2.47063       0.18781      173.0569     <.0001
      Games                 1       -2.86329       0.18802      231.9065     <.0001
```

The parameter estimates are arranged from most preferred to least preferred. These results are also available in a SAS data set called `parmest`.

You could change the reference level using the `classopts=` option as follows:

```
   %mktmdiff(bw, nattrs=9, nsets=18, setsize=5,
             attrs=attrlist, classopts=zero='Internet',
             data=bestworst, design=sasuser.bibd)
```

The new parameter estimates table is as follows:

```
                  Best Worst Example with Cell Phone Attributes
                      Multinomial Logit Parameter Estimates

                                Parameter      Standard
                         DF      Estimate         Error    Chi-Square    Pr > ChiSq

   Large Letters          1       2.44592       0.18731      170.5156        <.0001
   Battery Life           1       1.92583       0.18533      107.9789        <.0001
   Free Replacement       1       1.43363       0.18658       59.0424        <.0001
   Camera                 1       1.13741       0.18484       37.8650        <.0001
   Applications           1       0.51485       0.18056        8.1305        0.0044
   Flip                   1       0.43700       0.18045        5.8650        0.0154
   Hands Free             1      -0.02471       0.17655        0.0196        0.8887
   Games                  1      -0.41737       0.17375        5.7702        0.0163
```

Note that the parameter estimates all change by a constant amount. If you take the original estimate for the internet parameter and subtract it from all of the original estimates, you get the new estimates.

Experimental Design for a MaxDiff Study

The experimental design for a MaxDiff study is a block design. A set of t attributes are organized into b blocks or sets of size k. Ideally, we prefer to use a balanced incomplete block design or BIBD. In a BIBD, each of the t attributes appears the same number of times, and each of the t attributes appears with every other attribute the same number of times. However, BIBDs do not exist for every combination of b, t, and k that might be of interest. Hence, in some situations we are forced to use unbalanced block designs in which each of the t attributes appears the same number of times, but not all of the $t(t-1)/2$ pairs of attributes appear the same number of times. Fortunately, while a BIBD is nice, it is not required for a MaxDiff analysis.

There is a set of necessary but not sufficient conditions for the existence of a BIBD. See page 971. You can use the `%MktBSize` macro to get lists of designs that meet these conditions for ranges of b, t, and k, for example, as follows.

```
   %mktbsize(nattrs=1 to 20, setsize=2 to 0.5 * t, nsets=t to 100)
```

This step also shows that you can restrict the sizes being considered using the values of b, t, and k. For example, this step will never consider a set size more than one half the number attributes. Nor will it consider designs with fewer sets than attributes. The results of this step are not shown.

Then you can use the `%MktBIBD` macro to find either a BIBD or an unbalanced block design, for example, as follows:

```
   %mktbibd(nattrs=12, setsize=6, nsets=22, seed=104)
```

There is never a guarantee that the `%MktBIBD` macro will find a BIBD, even when one exists. It does a computerized search using PROC OPTEX. However, it does a good job of finding small BIBDs and coming very close for larger ones—certainly close enough for most MaxDiff studies. However, just because you run the %MktBIBD macro and get something back does not mean it is suitable for use. Like any design, you need to examine its properties and ensure that it meets your needs. Consider, for example, the following step, which creates a design with 20 attributes shown in 16 sets of size 5:

```
%mktbibd(nattrs=20, setsize=5, nsets=16, seed=292, out=sasuser.maxdiffdes)
```

The attribute by attribute frequency matrix is as follows:

```
                    Attribute by Attribute Frequencies

         1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20

    1    4   1   1   1   1   1   1   0   0   0   1   1   1   1   1   1   1   1   1   1
    2        4   1   1   1   0   1   1   1   1   0   1   1   1   1   1   0   1   1   1
    3            4   1   1   1   1   1   1   1   1   1   0   1   1   1   1   1   0   0
    4                4   1   1   0   1   1   1   1   1   1   0   0   1   1   1   1   1
    5                    4   1   1   1   1   1   1   0   1   1   1   0   1   0   1   1
    6                        4   1   1   1   1   0   1   1   1   1   1   0   1   1   1
    7                            4   1   1   1   1   1   1   0   0   1   1   1   1   1
    8                                4   0   0   1   1   1   1   1   1   1   1   1   1
    9                                    4   0   1   1   1   1   1   1   1   1   1   1
   10                                        4   1   1   1   1   1   1   1   1   1   1
   11                                            4   1   1   1   1   1   0   1   1   1
   12                                                4   1   1   1   0   1   0   1   1
   13                                                    4   1   1   1   1   1   0   0
   14                                                        4   0   1   1   1   1   1
   15                                                            4   1   1   1   1   1
   16                                                                4   1   0   1   1
   17                                                                    4   1   1   1
   18                                                                        4   1   1
   19                                                                            4   0
   20                                                                                4
```

The design (not shown) has $bk = 16 \times 5 = 80$ entries, and each of the 20 attributes appears exactly 4 times. However, the off-diagonal attribute by attribute frequencies are not all the same, so this is not a BIBD. Furthermore, some of the attributes never appear with other attributes, so this design is not a good candidate for a MaxDiff study.

The %MktBIBD macro also tries to optimize the attribute by position frequencies. For this design, they are as follows:

```
                 Attribute by Position Frequencies

                            1  2  3  4  5

                        1   1  1  0  1  1
                        2   0  1  1  1  1
                        3   1  1  0  1  1
                        4   1  1  1  1  0
                        5   1  0  1  1  1
                        6   1  1  1  0  1
                        7   1  1  0  1  1
                        8   1  1  1  0  1
                        9   1  0  1  1  1
                       10   0  1  1  1  1
                       11   1  1  1  1  0
                       12   1  1  1  0  1
                       13   1  1  0  1  1
                       14   1  1  1  0  1
                       15   0  1  1  1  1
                       16   1  1  1  1  0
                       17   1  1  1  1  0
                       18   1  0  1  1  1
                       19   1  0  1  1  1
                       20   0  1  1  1  1
```

Given the b, t, and k specifications, these frequencies are optimal. There are no 2's, and there is exactly one zero in each row, that is, each attribute appears in all but one position.

The following step requests 20 blocks or sets instead of 16:

```
   %mktbibd(nattrs=20, setsize=5, nsets=20, seed=292, out=sasuser.maxdiffdes)
```

The number of sets is increased by 4 since $4k = t = 20$. You have to increase b by multiples of 4 (given this k and t) so that each attribute can appear an equal number of times. The attribute by attribute frequency matrix is as follows:

```
                         Attribute by Attribute Frequencies

       1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20

  1    5  1  1  1  1  1  1  1  1  1  1  2  1  1  1  1  1  1  1  1
  2       5  1  1  1  1  2  1  1  1  1  1  1  1  1  1  1  1  1  1
  3          5  1  1  1  1  1  1  1  1  1  1  1  1  1  1  2  1  1
  4             5  1  1  1  1  1  1  1  1  1  2  1  1  1  1  1  1
  5                5  1  1  1  1  1  1  1  1  1  1  1  1  1  1  2
  6                   5  1  0  1  1  1  1  1  1  1  2  1  1  2  1
  7                      5  1  1  1  1  1  1  1  1  1  1  1  1  1
  8                         5  1  1  1  1  1  1  1  2  1  1  2  1
  9                            5  1  1  1  1  1  2  1  1  1  1  1
 10                               5  1  1  2  1  1  1  1  1  1  1
 11                                  5  1  1  1  1  1  2  1  1  1
 12                                     5  1  1  1  1  1  1  1  1
 13                                        5  1  1  1  1  1  1  1
 14                                           5  1  1  1  1  1  1
 15                                              5  1  1  1  1  1
 16                                                 5  1  1  0  1
 17                                                    5  1  1  1
 18                                                       5  1  1
 19                                                          5  1
 20                                                             5
```

Now each attribute appears 5 times, and the pairwise frequencies are all 1's and 2's with two 0's. Given the *b*, *t*, and *k* specifications, these frequencies are not quite what we hoped. There are more 2's than 0's, so ideally we would like the 0's to go away. The macro finds designs with this pattern of frequencies over and over, so it is likely that a design of this size with no zero frequencies is impossible. Block designs are subject to all sorts of Sudoku-like rules—if this treatment occurs with that treatment in this block, then it cannot occur with this other treatment in that other block. Every time PROC OPTEX tries to change a 0 frequency to a 1, it changes a 1 to a 0 rather than a 2 to a 1.

The attribute by position frequencies are as follows:

```
                    Attribute by Position Frequencies

                                1  2  3  4  5

                            1   1  1  1  1  1
                            2   1  1  1  1  1
                            3   1  1  1  1  1
                            4   1  1  1  1  1
                            5   1  1  1  1  1
                            6   1  1  1  1  1
                            7   1  1  1  1  1
                            8   1  1  1  1  1
                            9   1  1  1  1  1
                           10   1  1  1  1  1
                           11   1  1  1  1  1
                           12   1  1  1  1  1
                           13   1  1  1  1  1
                           14   1  1  1  1  1
                           15   1  1  1  1  1
                           16   1  1  1  1  1
                           17   1  1  1  1  1
                           18   1  1  1  1  1
                           19   1  1  1  1  1
                           20   1  1  1  1  1
```

These are perfect. Each attribute appears in every position exactly once.

The following step requests 24 blocks or sets instead of 16 or 20:

```
   %mktbibd(nattrs=20, setsize=5, nsets=24, seed=292, out=sasuser.maxdiffdes)
```

The attribute by attribute frequency matrix is as follows:

```
                        Attribute by Attribute Frequencies

       1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20

   1   6  1  1  1  1  1  1  1  1  2  2  1  2  1  2  1  1  1  2  1
   2      6  2  1  1  1  2  1  1  1  1  2  1  1  1  2  2  1  1  1
   3         6  1  2  2  1  2  1  1  1  1  1  1  1  1  1  2  1  1
   4            6  2  1  1  1  2  2  1  2  1  1  1  1  1  1  2  1
   5               6  1  2  1  1  1  1  1  2  1  1  1  1  1  1  2
   6                  6  1  1  1  1  1  1  1  1  1  2  2  1  2  2
   7                     6  1  2  2  1  1  1  2  1  1  1  1  1  1
   8                        6  1  1  2  1  1  2  1  2  2  1  1  1
   9                           6  1  1  1  2  1  1  1  1  2  1  2
  10                              6  1  1  1  1  1  2  2  1  1  1
  11                                 6  2  1  1  1  1  1  2  1  2
  12                                    6  1  1  2  1  1  1  2  1
  13                                       6  2  1  1  2  1  1  1
  14                                          6  2  1  1  1  2  1
  15                                             6  1  1  2  1  2
  16                                                6  1  2  1  1
  17                                                   6  1  1  1
  18                                                      6  1  1
  19                                                         6  1
  20                                                            6
```

Now each attribute appears 6 times, and the pairwise frequencies are all 1's and 2's with no 0's.

The attribute by position frequencies are as follows:

```
                        Attribute by Position Frequencies

                                 1  2  3  4  5

                             1   1  1  2  1  1
                             2   1  2  1  1  1
                             3   1  1  1  2  1
                             4   1  1  1  1  2
                             5   2  1  1  1  1
                             6   1  1  2  1  1
                             7   1  1  1  2  1
                             8   1  1  2  1  1
                             9   1  2  1  1  1
                            10   1  2  1  1  1
```

```
                                  11   1   1   1   2   1
                                  12   1   1   1   1   2
                                  13   2   1   1   1   1
                                  14   1   2   1   1   1
                                  15   2   1   1   1   1
                                  16   1   1   1   2   1
                                  17   2   1   1   1   1
                                  18   1   1   1   1   2
                                  19   1   1   1   1   2
                                  20   1   1   2   1   1
```

Again, given the *b*, *t*, and *k* specifications, these frequencies are optimal. There are no 0's or 3's, and there is exactly one 2 in each row, that is, each attribute appears in all positions but one more than the others. Often times, we are content to use a design with patterns of frequencies like we see here, not perfect, but optimal given the design specifications.

You could run the following step to see if a BIBD is possible if you change the block size:

```
   %mktbsize(nattrs=20, setsize=5, nsets=t to 500)
```

The results are as follows:

```
                                                                   n
         t              k       b          r            Lambda         Total
         Number of      Set     Number     Attribute    Pairwise       Sample
         Attributes     Size    of Sets    Frequency    Frequencies    Size

             20           5        76          19              4         380
```

You would need 76 sets before you would have any hope of finding a BIBD. If you are willing to change the number of messages or the number of messages shown at one time, then you have more possibilities. The following step investigates:

```
   %mktbsize(nattrs=18 to 22, setsize=4 to 6, nsets=t to 500)
```

The results are as follows:

```
                                                                   n
         t              k       b          r            Lambda         Total
         Number of      Set     Number     Attribute    Pairwise       Sample
         Attributes     Size    of Sets    Frequency    Frequencies    Size

             18           4       153          34              6         612
             18           5       306          85             20        1530
             18           6        51          17              5         306
```

```
           19           4        57          12             2          228
           19           5       171          45            10          855
           19           6        57          18             5          342
           20           4        95          19             3          380
           20           5        76          19             4          380
           20           6       190          57            15         1140
           21           4       105          20             3          420
           21           5        21           5             1          105
           21           6        28           8             2          168
           22           4        77          14             2          308
           22           5       462         105            20         2310
           22           6        77          21             5          462
```

We can try to find a BIBD with 21 attributes, and 21 sets of size 5 as follows:

```
%mktbibd(nattrs=21, setsize=5, nsets=21, seed=292, out=sasuser.maxdiffdes)
```

The attribute by attribute frequency matrix is as follows:

```
                    Attribute by Attribute Frequencies

      1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21

  1   5  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  2      5  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  3         5  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  4            5  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  5               5  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  6                  5  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  7                     5  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  8                        5  1  1  1  1  1  1  1  1  1  1  1  1  1
  9                           5  1  1  1  1  1  1  1  1  1  1  1  1
 10                              5  1  1  1  1  1  1  1  1  1  1  1
 11                                 5  1  1  1  1  1  1  1  1  1  1
 12                                    5  1  1  1  1  1  1  1  1  1
 13                                       5  1  1  1  1  1  1  1  1
 14                                          5  1  1  1  1  1  1  1
 15                                             5  1  1  1  1  1  1
 16                                                5  1  1  1  1  1
 17                                                   5  1  1  1  1
 18                                                      5  1  1  1
 19                                                         5  1  1
 20                                                            5  1
 21                                                               5
```

The attribute by position frequencies are as follows:

```
                    Attribute by Position Frequencies

                               1  2  3  4  5

                           1   1  1  1  1  1
                           2   1  1  1  1  1
                           3   1  1  1  1  1
                           4   1  1  1  1  1
                           5   1  1  1  1  1
                           6   1  1  1  1  1
                           7   1  1  1  1  1
                           8   1  1  1  1  1
                           9   1  1  1  1  1
                          10   1  1  1  1  1
                          11   1  1  1  1  1
                          12   1  1  1  1  1
                          13   1  1  1  1  1
                          14   1  1  1  1  1
                          15   1  1  1  1  1
                          16   1  1  1  1  1
                          17   1  1  1  1  1
                          18   1  1  1  1  1
                          19   1  1  1  1  1
                          20   1  1  1  1  1
                          21   1  1  1  1  1
```

These results are perfect. The art of design is determining what ranges of parameters work for you and then finding the best design that works for what you need.

%MktMDiff Macro Options

The following options can be used with the `%MktMDiff` macro:

Option	Description
`layout`	(positional) choice data set structure
	(positional) "help" or "?" displays syntax summary
`attrs=`*macro-variable*	macro variable with attribute descriptions
`b=`*b*	number of sets (alias for `nsets=`)
`classopts=`*options-list*	class variable options
`data=`*SAS-data-set*	input data set
`design=`*SAS-data-set*	design data set
`group=`*k*	number of groups in the `design=` data set
`k=`*k*	set size (alias for `setsize=`)
`nattrs=`*t*	number of attributes (alias for `t=`)

Option	Description
`nsets=`*b*	number of sets (alias for `b=`)
`options=nocode`	just create the OUT= data set
`options=noanalysis`	just create the OUT= and OUTCODED= data sets
`options=nosort`	do not sort the parameter estimates table
`options=nodrop`	do not drop the variable or parameter column
`options=rescale`	do `rescale=` even when dubious
`out=`*SAS-data-set*	merged data and design data set
`outcoded=`*SAS-data-set*	coded data set
`outparm=`*SAS-data-set*	parameter estimates data set
`rescale=`*value*	table of rescaled parameter estimates
`setsize=`*k*	set size (alias for `k=`)
`t=`*t*	number of attributes or messages (alias for `nattrs=`)
`vars=`*variable-list*	`data=` data set variables

Help Option

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%mktmdiff(help)
%mktmdiff(?)
```

Required Options

The `layout`, `b=` or `nsets=`, `k=` or `setsize=`, `t=` or `nattrs=`, and the `design=` options must be specified. The `layout` option is a positional parameter and must be specified first.

layout

specifies a nonoptional positional parameter that indicates the structure of the choice data. Just specify a value, not `layout=`. The value has several components, you can use any case, and spaces between the pieces are optional. Values include `b`, `w`, `alt`, and `pos`. Both a `b` and a `w` must be specified. When `b` comes before `w`, the best variables come before the worst variables. Otherwise, when `w` comes before `b`, the worst variables come before the best variables. You can optionally specify `alt` as well which means that the variables alternate (e.g. `b1 w1 b2 w2 b3 w3` where the `b` variables are the best variables and the `w` variables are the worst variables). Otherwise it is expected that all of one type appear then all of the other type. By default, the data are assumed to be the numbers of the attributes that were chosen (e.g. 1 to 6 when there are a total of `nattrs=6` attributes). If instead (say with `setsize=3`), the data are 1 to 3 indicating the position of the chosen attribute, then you must specify `pos` as well.

Examples:

`bw`
(or `b w`)
best then worst, e.g. `b1-b6 w1-w6` or even `x1-x12` (if $2 \times b = 12$). While the values need to be best then worst, the variable names do not need to reflect this. The data are attribute numbers.

`wb`
(or `w b`)
worst then best, e.g. `w1-w6 b1-b6` or even `x1-x12` (if $2 \times b = 12$). While the values need to be worst then best, the variable names do not need to reflect this. The data are attribute numbers.

`bwalt`
(or `b w alt` or `alt b w` and so on)
best then worst and alternating, e.g. `b1 w1 b2 w2 ...` or even `x1-x12` (if $2 \times b = 12$). While the values need to alternate best then worst, the variable names do not need to reflect this. The data are attribute numbers.

`wbalt`
(or `w b alt` or `alt w b` and so on)
worst then best and alternating, e.g. `w1 b1 w2 b2 ...` or even `x1-x12` (if $2 \times b = 12$). While the values need to alternate worst then best, the variable names do not need to reflect this. The data are attribute numbers.

`bwpos`
(or `b w alt` or `b pos w` and so on)
best then worst, and the data are positions.

`wbpos`
(or `w b pos` or `w pos b` and so on)
worst then best, and the data are positions.

`bwaltpos`
(or `b w alt pos` or `alt b pos w` and so on)
best then worst and alternating, and the data are positions.

`wbaltpos`
(or `w b alt pos` or `alt w pos b` and so on)
worst then best and alternating, and the data are positions.

design= *SAS-data-set*

specifies the data set with the design. It is typically a BIBD in the format produced by the `out=` option of the `%MktBIBD` macro. You must specify this option. All numeric variables are assumed to contain the BIBD, unless there is exactly one extra variable, and the first or last variable is called `Group`. In that case, the group variable is ignored.

nattrs= *t*
t= *t*

specifies the number of attributes or messages. The `nattrs=` and `t=` options are aliases. This option (in one of its two forms) must be specified. The “`t`” in `t=` stands for treatments and corresponds to typical BIBD notation.

nsets= *b*
b= *b*

specifies the number of sets. The `nsets=` and `b=` options are aliases. This option (in one of its two forms) must be specified. The “`b`” in `b=` stands for blocks and corresponds to typical BIBD notation.

setsize= *k*
k= *k*
specifies the number of attributes or messages shown at one time (in a set). The `setsize=` and `k=` options are aliases. This option (in one of its two forms) must be specified. The `k=` option is named using typical BIBD notation.

Data Set Options

The `design=` option is a required data set option that is listed in the previous section. The following are optional data set options.

data= *SAS-data-set*
specifies the input data set. By default, the last data set created is used.

out= *SAS-data-set*
specifies the output data set in a form ready for coding. The default is `out=maxdiff`.

outcoded= *SAS-data-set*
specifies the output data set that has been coded by PROC TRANSREG. The default is `outcoded=coded`.

outparm= *SAS-data-set* specifies the output data set with the parameter estimates table. The default is `outparm=parmest`. If the `rescale=` option is specified, the requested results are added to this data set.

Other Options

attrs= *macro-variable*
specifies the name of a macro variable that contains a comma delimited list of attribute descriptions. Alternatively, after the name you can specify a single nonblank delimiter character. Examples: `attrs=myattrlist`, `attrs=myattrs -`, (using a dash as a delimiter). If you do not specify this option, the default attribute names are “A1”, “A2”, and so on. The macro variable must have a name that does not match any of the macro parameter names (so you cannot specify `attrs=attrs` for example). The name also must not match any macro names used internally by the macro. In the unlikely event you specify an invalid name, an error is displayed. Most names, including any name that contains '`att`' that is not '`attrs`' or '`nattrs`' is always going to work.

classopts= *options-list*
specifies options to apply to the class variable, for example: `classopts=effects`. The class variable in the final analysis data set contains the descriptions of the attributes that are specified in the `attrs=` macro variable. The default is `classopts=zero=none`. A reference cell coding is used, and the reference level is displayed with a parameter estimate of zero. If you specify a null value (`classopts=`,) then the default reference cell coding is used and the reference level is not displayed. You can control the reference level by specifying `classopts=zero=`*'reference-level'* and naming the appropriate reference level. The reference level will exactly match one of the descriptions in the `attrs=` macro variable.

group= *g*

specifies the number of blocks of choice sets. By default, when `group=` is not specified, it is assumed that the design consists of one big group. Otherwise, with n subjects in each group and `group=`g there are $n \times g$ rows of data each consisting of b/g best values and b/g worst values. The number of subjects in each group must be the same. It is also assumed that the design is sorted by the group variable. If the design is made by the `%MktBIBD` macro, this will always be the case. If the `design=` data set has $k + 1$ variables and either the first variable or the last variable is called `Group` (case is ignored), then that variable is ignored and not treated as part of the design. Otherwise, the `design=` data set is required to have k variables.

options= *options-list*

specifies binary options. By default, none of these options are specified. Specify one or more of the following values after `options=`.

`nocode`
just create the OUT= data set but do not code or do the analysis.

`noanalysis`
just create the OUT= and OUTCODED= data sets but do not do the analysis.

`nosort`
do not sort the parameter estimates table by the parameter estimates.

`nodrop`
do not drop the variable or parameter column from the parameter estimates table.

`rescale`
ignore the usual restriction that `rescale=` option can only be used in the context of the default `classopts=`. If you use this option, you must ensure that you are only using the `zero=` option to change the reference level or are otherwise doing something that will not change the coding from reference cell to something else.

rescale= *value*

specifies various ways to rescale the parameter estimates. You can specify any of the following values:

`default`
ordinary parameter estimates.

`center`
parameter estimates are centered.

`p`
parameter estimates are scaled to probabilities:

$$\frac{\exp(\hat{\beta}_i)}{\sum_{j=1}^{m} \exp(\hat{\beta}_j)}$$

`p100`
parameter estimates are scaled to probabilities then multiplied by 100:

$$\frac{100 \exp(\hat{\beta}_i)}{\sum_{j=1}^{m} \exp(\hat{\beta}_j)}$$

`adjusted`
`adj`
parameter estimates are adjusted by the number of attributes in each set. First, $\hat{\boldsymbol{\beta}}$ is centered then scaled as follows:

$$\frac{\exp(\hat{\beta}_i)}{\exp(\hat{\beta}_i) + k - 1}$$

Finally, the adjusted values are rescaled to sum to 1.

`adjusted100`
`adj100`
parameter estimates are adjusted by the number of attributes in each set. First, $\hat{\boldsymbol{\beta}}$ is centered then scaled as follows:

$$\frac{\exp(\hat{\beta}_i)}{\exp(\hat{\beta}_i) + k - 1}$$

Finally, the adjusted values are rescaled to sum to 100.

`all`
the default and all rescaled values are reported.

Note that `rescale=` and `rescale=default` are equivalent. When this option is specified (or any option besides `rescale=default` is specified), an additional table is displayed with the rescaled parameter estimates. You can specify multiple values and get multiple columns in the results table. Example: `rescale=default adj100`. All specified rescalings are added to the `outparm=` data set. These rescalings were suggested by Sawtooth Software (2005, 2007).

Do not use both the `rescale=` option and the `classopts=` option unless you are only changing details of the reference cell coding. If you use these options together, you must ensure that you are only using the `zero=` option to change the reference level or are otherwise doing something that will not change the coding from reference cell to something else. In that case, you can specify `options=rescale` to allow the analysis to proceed.

vars= *variable-list*

specifies the variables in the `data=` data set that contain the data. There must be $2 \times b$ variables in this list (from `nsets=`b). The default is all numeric variables in the data set.

%MktMDiff Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktMerge Macro

The `%MktMerge` autocall macro merges a data set containing a choice design with choice data. See the following pages for examples of using this macro in the design chapter: 149 and 176. Also see the following pages for examples of using this macro in the discrete choice chapter: 325, 371, 387, 437, 522, and 529. Additional examples appear throughout this chapter. The following shows a typical example of using this macro:

```
%mktmerge(design=rolled, data=results, out=res2,
          nsets=18, nalts=5, setvars=choose1-choose18)
```

The `design=` data set comes from the `%MktRoll` macro. The `data=` data set contains the data, and the `setvars=` variables in the `data=` data set contain the numbers of the chosen alternatives for each of the 18 choice sets. The `nsets=` option specifies the number of choice sets, and the `nalts=` option specifies the number of alternatives. The `out=` option names the output SAS data set that contains the experimental design and a variable `c` that contains 1 for the chosen alternatives (first choice) and 2 for unchosen alternatives (second or subsequent choice).

When the `data=` data set contains a blocking variable, name it in the `blocks=` option. When there is blocking, it is assumed that the `design=` data set contains blocks of *nalts* × *nsets* observations. The `blocks=` variable must contain values 1, 2, ..., *n* for *n* blocks. The following example uses the `%MktMerge` macro with blocking:

```
%mktmerge(design=rolled, data=results, out=res2, blocks=form,
          nsets=18, nalts=5, setvars=choose1-choose18)
```

%MktMerge Macro Options

The following options can be used with the `%MktMerge` macro:

Option	Description
`help`	(positional) "help" or "?" displays syntax summary
`blocks=`1 | *variable*	blocking variable
`data=`*SAS-data-set*	input SAS data set
`design=`*SAS-data-set*	input SAS choice design data set
`nalts=`*n*	number of alternatives
`nsets=`*n*	number of choice sets
`out=`*SAS-data-set*	output SAS data set
`setvars=`*variable-list*	variables with the data
`statements=`*SAS-statements*	additional statements

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%mktmerge(help)
%mktmerge(?)
```

You must specify the `design=`, `nalts=`, `nsets=`, and `setvars=` options.

blocks= 1 | *variable*
specifies either a 1 (the default) if there is no blocking or the name of a variable in the `data=` data set that contains the block number. When there is blocking, it is assumed that the `design=` data set contains blocks of *nalts* × *nsets* observations, one set per block. The `blocks=` variable must contain values 1, 2, ..., *n* for *n* blocks.

data= *SAS-data-set*
specifies an input SAS data set with data for the choice model. By default, the `data=` data set is the last data set created.

design= *SAS-data-set*
specifies an input SAS data set with the choice design. This data set could have been created, for example, with the `%MktRoll` or `%ChoicEff` macros. This option must be specified.

nalts= *n*
specifies the number of alternatives. This option must be specified.

nsets= *n*
specifies the number of choice sets. This option must be specified.

out= *SAS-data-set*
specifies the output SAS data set. If `out=` is not specified, the DATA*n* convention is used. This data set contains the experimental design and a variable `c` that contains 1 for the chosen alternatives (first choice) and 2 for unchosen alternatives (second or subsequent choice).

setvars= *variable-list*
specifies a list of variables, one per choice set, in the `data=` data set that contains the numbers of the chosen alternatives. It is assumed that the values of these variables range from 1 to *nalts*. This option must be specified.

statements= SAS-statements
specifies additional statements like `format` and `label` statements. This option is illustrated in the following step:

```
%mktmerge(design=rolled, data=results, out=res2, blocks=form,
          nsets=&n, nalts=&m, setvars=choose1-choose&n,
          statements=%str(price = input(put(price, price.), 5.);
                          format scene scene. lodge lodge.;))
```

%MktMerge Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktOrth Macro

The %MktOrth autocall macro lists some of the 100% orthogonal main-effects plans that the %MktEx macro can generate. See page 106 for an example of using this macro in the design chapter. Also see the following pages for examples of using this macro in the discrete choice chapter: 342 and 660. Additional examples appear throughout this chapter.

Mostly, you use this macro indirectly; it is called by the %MktEx macro. However, you can directly call the %MktOrth macro to see what orthogonal designs are available and decide which ones to use. The following step requests all the designs in the catalog with 100 or fewer runs and two-level through six-level factors (with no higher-level factors.)

```
   %mktorth(maxn=100, maxlev=6)
```

The macro creates data sets and displays no output except the following notes:

```
   NOTE: The data set WORK.MKTDESLEV has 347 observations and 9 variables.
   NOTE: The data set WORK.MKTDESCAT has 347 observations and 3 variables.
```

This next step generates the entire catalog of 119,852 designs* including over 62,000 designs in 512 runs that are not generated by default:

```
   %mktorth(maxlev=144, options=512)
```

This step might take on the order of several minutes to run.

This next step generates the catalog of approximately 57 thousand designs including designs with up to 144-level factors:

```
   %mktorth(maxlev=144)
```

This step might take on the order of several minutes to run. Unless you really want to see all of the designs, you can make the %MktOrth macro run much faster by specifying smaller values for range= or maxn= (which control the number of runs) and maxlev= (which controls the maximum number of factor levels and the number of variables in the MKTDESLEV data set) than the defaults (range=n le 1000, maxn=1000, maxlev=50). The maximum number of levels you can specify is 144.

The following step lists the first few and the last few designs in the catalog:

```
   proc print data=mktdeslev(where=(n le 12 or n ge 972));
      var design reference;
      id n; by n;
      run;
```

*Elsewhere in this chapter, the size of the orthogonal array catalog is reported to be 117,556. The discrepancy is due to the 2296 designs that are explicitly in the catalog and have more than 513 runs. Most are constructed from the parent array 24^8 in 576 runs (which is useful for making Latin Square designs). The rest are constructed from Hadamard matrices.

Some of the results are as follows:

```
      n              Design                Reference

      4    2 **  3                         Hadamard

      6    2 **  1  3 **  1                Full-Factorial

      8    2 **  7                         Hadamard
           2 **  4            4 **  1      Fractional-Factorial

      9             3 **  4                Fractional-Factorial

     10    2 **  1            5 **  1      Full-Factorial

     12    2 ** 11                         Hadamard
           2 **  4  3 **  1                Orthogonal Array
           2 **  2            6 **  1      Orthogonal Array
                    3 **  1   4 **  1      Full-Factorial

    972    2 **971                         Hadamard

    976    2 **975                         Hadamard
           2 **972            4 **  1      Orthogonal Array
           2 **969            4 **  2      Orthogonal Array
           2 **968            8 **  1      Orthogonal Array
           2 **966            4 **  3      Orthogonal Array

    984    2 **983                         Hadamard
           2 **980            4 **  1      Orthogonal Array

    992    2 **991                         Hadamard
           2 **988            4 **  1      Orthogonal Array
           2 **985            4 **  2      Orthogonal Array
           2 **984            8 **  1      Orthogonal Array
           2 **982            4 **  3      Orthogonal Array
           2 **979            4 **  4      Orthogonal Array
           2 **976            4 **  5      Orthogonal Array
           2 **976           16 **  1      Orthogonal Array
           2 **973            4 **  6      Orthogonal Array
           2 **970            4 **  7      Orthogonal Array

   1000    2 **999                         Hadamard
           2 **996            4 **  1      Orthogonal Array
```

In most ways, the catalog stops at 513 runs. The exceptions include: 24^8 in 576 runs, Hadamard designs up to $n = 1000$, and designs easily constructed from those Hadamard designs (by creating, 4, 8, 16, and so on level factors).

The following step displays the first few designs and variables in the MKTDESLEV data set:

```
proc print data=mktdeslev(where=(n le 12));
   var design reference x1-x6;
   id n; by n;
   run;
```

Some of the results are as follows:

```
    n           Design           Reference            x1  x2  x3  x4  x5  x6

    4  2 **  3                   Hadamard              0   3   0   0   0   0

    6  2 **  1  3 **  1          Full-Factorial        0   1   1   0   0   0

    8  2 **  7                   Hadamard              0   7   0   0   0   0
       2 **  4            4 **  1 Fractional-Factorial  0   4   0   1   0   0

    9           3 **  4          Fractional-Factorial  0   0   4   0   0   0

   10  2 **  1            5 **  1 Full-Factorial        0   1   0   0   1   0

   12  2 ** 11                   Hadamard              0  11   0   0   0   0
       2 **  4  3 **  1          Orthogonal Array      0   4   1   0   0   0
       2 **  2            6 **  1 Orthogonal Array      0   2   0   0   0   1
                3 **  1   4 **  1 Full-Factorial        0   0   1   1   0   0
```

If you just want to display a list of designs, possibly selecting on n, the number of runs, you can use the MKTDESCAT data set. However, if you would like to do more advanced processing, based on the numbers of levels of some of the factors, you can use the `outlev=mktdeslev` data set to select potential designs. You can look at the level information in MKTDESLEV and see the number of two-level factors in `x2`, the number of three-level factors in `x3`, ..., the number of fifty-level factors is in `x50`, ..., and the number of 144-level factors in `x144`. The number of one-level factors, `x1`, is always zero, but `x1` is available so you can make arrays (for example, `array x[50]`) and have `x[2]` refer to `x2`, the number of two-level factors, and so on.

Say you are interested in the design $2^5 3^5 4^1$. The following steps display some of the ways in which it is available:

```
%mktorth(maxn=100)

proc print data=mktdeslev noobs;
   where x2 ge 5 and x3 ge 5 and x4 ge 1;
   var n design reference;
   run;
```

Some of the results are as follows:

```
  n                   Design                          Reference

 72    2 ** 44  3 ** 12   4 **  1                  Orthogonal Array
 72    2 ** 43  3 **  8   4 **  1   6 **  1        Orthogonal Array
 72    2 ** 37  3 ** 13   4 **  1                  Orthogonal Array
 72    2 ** 36  3 **  9   4 **  1   6 **  1        Orthogonal Array
 72    2 ** 35  3 ** 12   4 **  1   6 **  1        Orthogonal Array
 .
 .
 .
```

The following steps illustrate one way that you can see all of the designs in a certain range of sizes:

```
%mktorth(range=12 le n le 20)

proc print; id n; by n; run;
```

The results are as follows:

```
  n                   Design                     Reference

 12    2 ** 11                                   Hadamard
       2 **  4  3 **  1                          Orthogonal Array
       2 **  2            6 **  1                Orthogonal Array
                3 **  1   4 **  1                Full-Factorial

 14    2 **  1            7 **  1                Full-Factorial

 15             3 **  1   5 **  1                Full-Factorial

 16    2 ** 15                                   Hadamard
       2 ** 12            4 **  1                Fractional-Factorial
       2 **  9            4 **  2                Fractional-Factorial
       2 **  8            8 **  1                Fractional-Factorial
       2 **  6            4 **  3                Fractional-Factorial
       2 **  3            4 **  4                Fractional-Factorial
                          4 **  5                Fractional-Factorial

 18    2 **  1  3 **  7                          Orthogonal Array
       2 **  1            9 **  1                Full-Factorial
                3 **  6   6 **  1                Orthogonal Array
```

```
        20    2 ** 19                                   Hadamard
              2 **  8           5 **  1                 Orthogonal Array
              2 **  2          10 **  1                 Orthogonal Array
                                4 **  1   5 **  1       Full-Factorial
```

The %MktOrth macro can output the lineage of each design, which is the set of steps that the %MktEx macro uses to create it. The following steps illustrate this option:

```
%mktorth(range=n=36, options=lineage)

proc print noobs;
   where index(design, '2 ** 11') and index(design, '3 ** 12');
   run;
```

The results are as follows:

```
                  n         Design            Reference

                 36    2 ** 11  3 ** 12    Orthogonal Array

                                        Lineage

                 36 ** 1 : 36 ** 1 > 3 ** 12 12 ** 1 : 12 ** 1 > 2 ** 11
```

The design $2^{11}3^{12}$ in 36 runs starts out as a single 36-level factor, 36^1. Then 36^1 is replaced by $3^{12}12^1$. Finally, 12^1 is replaced by 2^{11} resulting in $2^{11}3^{12}$.

%MktOrth Macro Options

The following options can be used with the %MktOrth macro:

Option	Description
help	(positional) "help" or "?" displays syntax summary
filter=*n*	extra filtering of the design catalog
maxn=*n*	maximum number of runs of interest
maxlev=*n*	maximum number of levels
options=lineage	construct the design lineage
options=mktex	the macro is being called from the %MktEx macro
options=mktruns	the macro is being called from the %MktRuns macro
options=parent	lists only parent designs
options=dups	suppress duplicate and inferior design filtering
options=512	adds some designs in 512 runs
outall=*SAS-data-set*	output data set with all designs
outcat=*SAS-data-set*	design catalog data set
outlev=*SAS-data-set*	output data set with the list of levels
range=*range-specification*	number of runs of interest

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%mktorth(help)
%mktorth(?)
```

filter= *n*

specifies extra design catalog filtering. Usually, you will never have to use this option. By default, `%MktOrth` filters out inferior designs. On the one hand, this process is expensive, but it also saves some time and resources by limiting the information that must be processed. Care is taken to do only the minimum amount of work to filter. However, this gets complicated. If you specify `maxlev=144`, the maximum, you will get perfect filtering of duplicates in a reasonable amount of time. `%MktOrth` uses internal and optimized constants to avoid doing extra work. Unfortunately, these constants might not be optimal for any other `maxlev=` value. This is almost never going to be a real issue. If however, you want to specify both `maxlev=` and ensure you get better filtering, specify `filter=`*n* (for some *n*) and you might get a few more inferior designs filtered out. The *n* value increases how deeply into the catalog `%MktOrth` searches for inferior designs. Larger values find more designs to exclude at a cost of greater run time. The following steps use both the default filtering and specify `filter=1000`:

```
%mktorth(maxlev=20)
%mktorth(maxlev=20, filter=1000)
```

The latter specification removes on the order of thirty more designs but at cost of much slower run time. Actually, `filter=27` is large enough for this example, and it has a negligible effect on run time, but you have to run the macro multiple times to figure that out. Values of a couple hundred or so are probably always going to be sufficient.

maxlev= *n*

specifies the maximum number of levels to consider. Specify a value *n*, such that $2 \le n \le 144$. The default is `maxlev=50`. This option controls the number of `x` variables in the `outlev=` data set. It also excludes from consideration designs with factors of more than `maxlev=` levels so it affects the number of rows in the output data sets. Note that specifying `maxlev=`*n* does not preclude designs with more than *n*-level factors from being used as parents for other designs, it just precludes the larger designs from being output. For example, with `maxlev=3`, the design $3^{12}12^1$ in 36 runs is used to make $2^{11}3^{12}$ before the $3^{12}12^1$ design is discarded. Specifying smaller values will make the macro run faster. With the maximum, `maxlev=144`, run time to generate the entire catalog can be on the order of several minutes.

maxn= *n*

specifies the maximum number of runs of interest. Specifying small numbers (e.g. $n \le 200$) will make the macro run faster.

options= *options-list*

specifies binary options. By default, none of these options are specified. Specify one or more of the following values after `options=`.

`lineage`
construct the design lineage, which is the set of instructions on how the design is made.

`mktex`
specifies that the macro is being called from the `%MktEx` macro and just the `outlev=` data set is needed. The macro takes short cuts to make it run faster doing only what the `%MktEx` macro needs.

`mktruns`
specifies that the macro is being called from the `%MktRuns` macro and just the `outlev=` data set is needed. The macro takes short cuts to make it run faster doing only what `%MktRuns` needs.

`parent`
specifies that only parent designs should be listed.

`dups`
specifies that the `%MktRuns` macro should not filter out duplicate and inferior designs from the catalog. This can be useful when you are creating a data set for the `cat=` option in the `%MktEx` macro.

`512`
adds some larger designs in 512 runs with mixes of 16, 8, 4, and 2-level factors to the catalog, which gives added flexibility in 512 runs at a cost of much slower run time. This option replaces the default parent design $4^{160}32^1$ with $16^{32}32^1$.

outall= *SAS-data-set*

specifies the output data set with all designs. This data set is not created by default. This data set is like the `outlev=` data set, except larger. The `outall=` data set includes *all* of the `%MktEx` design catalog, including all of the smaller designs that can be trivially made from larger designs by dropping factors. For example, when the `outlev=` data set has `x2`=2 `x3`=2, then the `outall=` data set has that design and also `x1=2 x3=1`, `x1=1 x3=2`, and `x1=1 x2=1`. When you specify `outall=` you must also specify a reasonably small `range=` or `maxn=` value. Otherwise, the `outall=` specification will take a *long* time and create a *huge* data set, which will very likely be too large to store on your computer.

outcat= *SAS-data-set*

specifies the output data set with the catalog of designs that the `%MktEx` macro can create. The default is `outcat=MktDesCat`.

outlev= *SAS-data-set*

specifies the output data set with the list of designs and 50 (by default) more variables, `x1-x50`, which includes: `x2` – the number of two-level factors, `x3` – the number of three-level factors, and so on. The default is `outlev=MktDesLev`. The number of `x` variables is determined by the `maxlev=` option.

range= *range-specification*

specifies the number of runs of interest. Specify a range involving `n`, where `n` is the number of runs. Your range specification must be a logical expression involving `n`. Examples:
`range=n=36`

```
range=18 le n le 36
range=n eq 18 or n eq 36
```

%MktOrth Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

The Orthogonal Array Catalog

This section contains information about the orthogonal array catalog. While this information is interesting and sometimes useful, this section can be skipped by most readers.

The `%MktOrth` macro maintains the orthogonal array catalog that the `%MktEx` and `%MktRuns` macros use. The `%MktOrth` macro instructs the `%MktEx` macro on both what orthogonal arrays exist and how to make them. In most cases, the `%MktOrth` macro is called by the `%MktEx` and `%MktRuns` macros and you never have to worry about it. However, you can use the `%MktOrth` macro to find out what designs exist in the orthogonal array catalog. This can be particularly useful for the orthogonal array specialist who seeks to understand the catalog and find candidates for undiscovered orthogonal arrays. Most designs explicitly appear in the catalog. Others are explicitly entered into the catalog, but they normally do not appear. Still others do not explicitly appear in the catalog, but the `%MktEx` macro can still figure out how to make them. These points and other aspects of the orthogonal array catalog are discussed in this section.

First, consider a design that is explicitly in the catalog. The following steps create a list of orthogonal arrays in 18 runs, display that list, then create the design using the `%MktEx` macro:

```
   %mktorth(range=n=18, options=lineage)

   data _null_;
      set;
      design = compbl(design);
      put 'Design:  ' design / 'Lineage: ' lineage /;
      run;

   %mktex(6 3 ** 6, n=18)
```

Of course the first two steps are not needed if your only goal is to make the design. The results are as follows:

```
Design:  2 ** 1 3 ** 7
Lineage: 18 ** 1 : 18 ** 1 > 3 ** 6 6 ** 1 : 6 ** 1 > 2 ** 1 3 ** 1

Design:  2 ** 1 9 ** 1
Lineage: 18 ** 1 : 18 ** 1 > 2 ** 1 9 ** 1 (parent)

Design:  3 ** 6 6 ** 1
Lineage: 18 ** 1 : 18 ** 1 > 3 ** 6 6 ** 1 (parent)
```

Three designs are listed. The third is the parent array $3^6 6^1$. The first is a child array $2^1 3^7$, which is constructed from the parent array $3^6 6^1$ by replacing the six-level factor with a two-level factor and a three-level factor. This can be seen in the lineage, which is explained in more detail later in this section. The second array is the full-factorial design $2^1 9^1$. All three of these arrays are explicitly in the catalog and can be directly produced by the `%MktEx` macro.

The catalog actually contains one more array that is created, but by default, it is discarded as inferior before the catalog is output or before `%MktEx` can use the catalog. You can use the `dups` option to see the duplicate and inferior arrays that would normally be discarded. The following steps illustrate this option:

```
%mktorth(range=n=18, options=lineage dups)

data _null_;
   set;
   design = compbl(design);
   put 'Design:  ' design / 'Lineage: ' lineage /;
   run;
```

The results are as follows:

```
Design:  2 ** 1 3 ** 7
Lineage: 18 ** 1 : 18 ** 1 > 3 ** 6 6 ** 1 : 6 ** 1 > 2 ** 1 3 ** 1

Design:  2 ** 1 3 ** 4
Lineage: 18 ** 1 : 18 ** 1 > 2 ** 1 9 ** 1 : 9 ** 1 > 3 ** 4

Design:  2 ** 1 9 ** 1
Lineage: 18 ** 1 : 18 ** 1 > 2 ** 1 9 ** 1 (parent)

Design:  3 ** 6 6 ** 1
Lineage: 18 ** 1 : 18 ** 1 > 3 ** 6 6 ** 1 (parent)
```

The new array is the second array in the list, $2^1 3^4$. It is made from the full-factorial design, $2^1 9^1$ by replacing the nine-level factor with 4 three-level factors. The resulting array, $2^1 3^4$, is inferior to $2^1 3^7$ in that the former has three fewer three-level factors than the latter. Hence, by default, it is discarded. This does not imply, however, that the $2^1 3^4$ design is a simple subset of the larger design, $2^1 3^7$. The smaller design might or might not be a subset, but typically it will not be. By default, the %MktEx macro operates under the assumption that orthogonal and balanced factors are interchangeable, so it always prefers arrays with more factors to arrays with fewer factors. By default, it will never produce the array $2^1 3^4$ that is based on the full-factorial design, $2^1 9^1$. It will always create $2^1 3^4$ from the first five columns of $2^1 3^7$.

The %MktEx macro provides you with a way to get these designs that are automatically excluded from the catalog. More generally, when there are multiple designs that meet your criteria, it gives you a way to explicitly choose which design you get. All you have to do is run the %MktOrth macro yourself, select which design you want, and feed just that one line of the catalog into the %MktEx macro. The following steps illustrate this option:

```
%mktorth(range=n=18, options=lineage dups)

data lev;
   set mktdeslev;
   where lineage ? '3 ** 4';
   run;

%mktex(2 3 ** 4,                    /* 1 two-level and 4 three-level factors*/
       n=18,                        /* 18 runs                              */
       cat=lev,                     /* OA catalog comes from lev data set   */
       out=alternative)             /* name of output design                */

%mktex(2 3 ** 4, n=18, out=default)
```

The where clause in the DATA step selects only one of the arrays since the other arrays that have more than 4 three-level factors have '3 ** 6' not '3 ** 4' in their lineage.

The default and the alternative (or “inferior”) arrays are displayed next:

Default Array					Alternative Array				
x1	x2	x3	x4	x5	x1	x2	x3	x4	x5
1	1	1	1	1	1	1	1	1	1
1	1	2	1	3	1	1	2	3	2
1	1	3	2	3	1	1	3	2	3
1	2	1	3	1	1	2	1	3	3
1	2	2	2	2	1	2	2	2	1
1	2	3	2	1	1	2	3	1	2
1	3	1	3	2	1	3	1	2	2
1	3	2	1	2	1	3	2	1	3
1	3	3	3	3	1	3	3	3	1
2	1	1	2	2	2	1	1	1	1
2	1	2	3	1	2	1	2	3	2
2	1	3	3	2	2	1	3	2	3
2	2	1	1	3	2	2	1	3	3
2	2	2	3	3	2	2	2	2	1
2	2	3	1	2	2	2	3	1	2
2	3	1	2	3	2	3	1	2	2
2	3	2	2	1	2	3	2	1	3
2	3	3	1	1	2	3	3	3	1

If only the first three factors had been requested, the two arrays would have been the same. This is because the first three factors form a full-factorial design. The remaining factors are different in the two arrays. Both are orthogonal and balanced, however, they differ in terms of their aliasing structure. In other words, they differ in terms of which effects are confounded. The GLM procedure can be used to examine the aliasing structure of a design. It provides a list of be estimable functions. Since GLM is a modeling procedure, it requires a dependent variable. Hence, the first step is to add a dependent variable y, which can be anything for our purposes, to each design. The following steps display the aliasing structure for main effects only:

```
data d;
   set default;
   y = 1;
   run;

data i;
   set alternative;
   y = 1;
   run;

proc glm data=d;
   ods select galiasing;
   model y = x1-x5 / e aliasing;
   run; quit;

proc glm data=i;
   ods select galiasing;
   model y = x1-x5 / e aliasing;
   run; quit;
```

The results for the default design are as follows:

```
                  General Form of Aliasing Structure

                              Intercept
                              x1
                              x2
                              x3
                              x4
                              x5
```

The results for the alternative design are as follows:

```
                  General Form of Aliasing Structure

                              Intercept
                              x1
                              x2
                              x3
                              x4
                              x5
```

The two results are the same, and they show that all effects can be estimated.

Next, we will try the same thing, but this time adding two-way interactions to the model. The following steps illustrate:

```
proc glm data=d;
   ods select galiasing;
   model y = x1-x5 x1|x2|x3|x4|x5@2 / e aliasing;
   run; quit;

proc glm data=i;
   model y = x1-x5 x1|x2|x3|x4|x5@2 / e aliasing;
   run; quit;
```

Note that the `x1-x5` list could be omitted from the independent variable specification, but leaving it in serves to list the main-effect terms first. The results for the default design are as follows:

```
                    General Form of Aliasing Structure

                                Intercept
                                x1
                                x2
                                x3
                                x4
                                x5
                                x1*x2
                                x1*x3
                                x2*x3
                                x1*x4
                                x2*x4
                                x3*x4
                                x1*x5
                                x2*x5
                                x3*x5
                                x4*x5
```

All main effects and two-factor interactions are estimable.

The results for the alternative design are as follows:

```
                    General Form of Aliasing Structure

                  Intercept - 2*x2*x5 - 2*x3*x5 - 2*x4*x5
                  x1
                  x2 + 2*x2*x5 + 0.5*x3*x5 - 1.5*x4*x5
                  x3 - 1.5*x2*x5 + 2*x3*x5 + 0.5*x4*x5
                  x4 + 0.5*x2*x5 - 1.5*x3*x5 + 2*x4*x5
                  x5 + 2*x2*x5 + 2*x3*x5 + 2*x4*x5
                  x1*x2
                  x1*x3
                  x2*x3 + 0.5*x2*x5 - 0.5*x3*x5
                  x1*x4
                  x2*x4 - 0.5*x2*x5 + 0.5*x4*x5
                  x3*x4 + 0.5*x3*x5 - 0.5*x4*x5
                  x1*x5
```

The first estimable function is a function of the intercept and 3 two-way interactions. In other words, the intercept is confounded with 3 of the two-way interactions. We can estimate the intercept if the two-way interactions are zero or negligible. The second estimable function is the first factor, x1. It is not confounded with any other effect in the model. The third estimable function is a combination of the second factor, x2 and 3 two-way interactions. In most cases, lower-order terms are confounded with higher-order terms. Both sets of results assume that all three-way and higher-way interactions are zero, since they are not specified in the model. You can add all interactions to the models as follows:

```
proc glm data=d;
   ods select galiasing;
   model y = x1-x5 x1|x2|x3|x4|x5@5 / e aliasing;
   run; quit;

proc glm data=i;
   model y = x1-x5 x1|x2|x3|x4|x5@5 / e aliasing;
   run; quit;
```

The first two terms from the default design are as follows:

```
Intercept + 27*x1*x5 + 37.75*x2*x5 + 120.5*x1*x2*x5 + 45*x1*x3*x5 +
62.917*x2*x3*x5 + 200.83*x1*x2*x3*x5 + 9*x4*x5 + 63*x1*x4*x5 + 97.083*x2*x4*x5 +
269.17*x1*x2*x4*x5 + 33*x3*x4*x5 + 129*x1*x3*x4*x5 + 202.58*x2*x3*x4*x5 +
522.17*x1*x2*x3*x4*x5

x1 - 13.5*x1*x5 - 28.75*x2*x5 - 80*x1*x2*x5 - 22.5*x1*x3*x5 - 47.917*x2*x3*x5 -
133.33*x1*x2*x3*x5 - 22.5*x1*x4*x5 - 61.083*x2*x4*x5 - 159.67*x1*x2*x4*x5 -
6*x3*x4*x5 - 43.5*x1*x3*x4*x5 - 124.58*x2*x3*x4*x5 - 307.67*x1*x2*x3*x4*x5
```

Again, lower-order terms are confounded with higher-order terms. This is the nature of using anything less than a full-factorial design. You typically assume that higher-order interactions are zero or negligible and hope for the best.

At least in terms of the aliasing structure and the two-way interactions, it appears that the default design is better than the alternative design. There are no guarantees, and if the aliasing structure is ever really important to you, you should consider and evaluate the other arrays in the catalog.

The %MktOrth macro maintains a very large design catalog. However, the number of possible orthogonal arrays is infinite, and even for finite n, the total number starts getting very large after $n = 143$. Hence, it is impossible for the catalog to be complete. Still, there are some orthogonal designs are not in the %MktOrth catalog that the %MktEx macro still knows how to make. They are in some sense larger than the arrays that exist in the catalog, but they have a regular enough structure that they can be created without an explicit lineage from the catalog. This is illustrated in the following steps:

```
%mktorth(range=n=12*13, options=lineage)

proc print; run;

%mktex(12 13, n=12*13)
```

The catalog has only one design in $12 \times 13 = 156$ runs, and it is a Hadamard matrix with 155 two-level factors. The catalog entry is as follows:

```
                    Obs     n     Design     Reference          Lineage

                      1    156    2 **155    Hadamard     2 ** 155 (parent)
```

Still, the %MktEx macro can construct a full-factorial design with a twelve-level and thirteen-level factor. The following steps show the part of the catalog that contains full-factorial designs:

```
%mktorth;

proc print; where reference ? 'Full'; run;
```

The largest n in the results (not shown) is 143. The goal is for the catalog to have complete coverage up to 143 runs and good coverage beyond that. Full-factorial designs beyond 143 runs are not needed for making child arrays, so they are not included. However, %MktEx is capable of recognizing and making many full-factorial designs with more than 143 runs. The following steps provide another example:

```
%mktorth(range=n=1008, options=lineage)

%mktex(2 ** 1007, n=1008)
```

There are no designs with 1008 runs in the catalog, but the %MktEx macro recognizes that this as a Hadamard design from the Paley 1 family, and makes it using the same code that makes smaller Hadamard matrices. (More precisely, it recognizes 1008 as 4×252, and it recognizes $252 - 1$ as prime and makable with the Paley 1 construction. The final design is the Kronecker product of Hadamard matrices of order 4 and of order 252.)

There is one more class of designs that the %MktEx macro can find even when they are not in the catalog. In the following example, the %MktEx macro finds the design 29^{30} in 841 runs:

```
%mktorth(range=n=29 * 29, options=lineage)

%mktex(29 ** 30, n=29*29)
```

This is a regular fractional-factorial design (the number of runs and all factor levels are a power of the same prime, 29) that PROC FACTEX (which the `%MktEx` macro calls) can find even though it is not in the `%MktOrth` catalog.

Similarly, in 256 runs, there are designs where the factor levels are powers of two that are not in the catalog, yet the `%MktEx` macro can make them. This is illustrated in the following steps:

```
%mktorth(range=n=256, options=lineage, maxlev=64)

proc print data=mktdeslev; where x64 and x4 ge 10; run;

%mktex(4 ** 10 64, n=256)
```

The results of the PROC PRINT step (not shown) show that the design $4^{10}64^1$ is not in the design catalog. Nevertheless, the tools in the `%MktEx` macro (specifically, PROC FACTEX) succeed in constructing this design. (Note that mixes of 2, 4, 8, and 16 level-factors in 256 runs are completely covered by the catalog.)

In 128 runs, most designs are created directly by the `%MktEx` macro using information in the design lineage. However, some, even though they are explicitly in the catalog, are created by PROC FACTEX without using the design lineage from the catalog.

Next, we will return to the topic of design lineage. The design lineage is a set of instructions for making a design with smaller level factors from a design with higher-level factors. Previously, we saw the following design in 18 runs:

```
Design:  2 ** 1 3 ** 7
Lineage: 18 ** 1 : 18 ** 1 > 3 ** 6 6 ** 1 : 6 ** 1 > 2 ** 1 3 ** 1
```

It starts in the `%MktEx` macro as a single 18-level factor. That is what the first part of the lineage, '`18 ** 1`' specifies. Then the 18-level factor is replaced by 6 three-level factors and one six-level factors. This is specified by the lineage fragment: `18 ** 1 > 3 ** 6 6 ** 1`. Finally, the six-level factor is replaced by a two-level factor and a three level factor: `6 ** 1 > 2 ** 1 3 ** 1`. The code that makes $3^6 6^1$ in 18 runs is the same code that replaces an 18-level factor in the design $3^{18}18^1$ in 54 runs, or replaces an 18-level factor in 72, 108, or more runs.

The following steps show the lineage for the design $2^7 4^5 8^{15}$ in 128 runs:

```
%mktorth(range=n=128, options=lineage)

data _null_;
   set mktdeslev;
   if x2 eq 7 and x4 eq 5 and x8 eq 15;
   design = compbl(design);
   put design / lineage /;
   run;
```

The design is shown on the first line of the following output and the lineage on the second:

```
2 ** 7 4 ** 5 8 ** 15
8 ** 16 16 ** 1 : 16 ** 1 > 4 ** 5 : 8 ** 1 > 2 ** 4 4 ** 1 : 4 ** 1 > 2 ** 3
```

The parent array is $8^{16}16^{1}$ not a single 128-level factor. The starting point is a single n-level factor for all arrays under 145 runs with the exception of some in 128 runs. Then the sixteen-level factor is replaced by 5 four-level factors. One of the eight-level factors is replaced by $2^{4}4^{1}$. A four-level factor is replaced by 3 two-level factors.

The following steps show a lineage for the design 2^{31} in 32 runs:

```
%mktorth(range=n=32, options=lineage)

data _null_;
   set mktdeslev;
   if x2 eq 31;
   design = compbl(design);
   put design / lineage /;
   run;
```

The design is shown on the first line of the following output and the lineage on the second:

```
2 ** 31
32 ** 1 : 32 ** 1 > 4 ** 8 8 ** 1 : 8 ** 1 > 2 ** 4 4 ** 1 : 4 ** 1 > 2 ** 3
```

This shows that the design starts as a single 32-level factor. It is replaced by $4^{8}8^{1}$. Next, 8^{1} is replaced by $2^{4}4^{1}$. Finally, 4^{1} is replaced by 2^{3}. Implicit in these instructions is the fact that substitutions can occur more than once. In this case, every four-level factor is replaced by 3 two-level factors. While multiple substitutions can occur for higher-level factors, in practice, multiple substitutions usually occur for only four-level factors and sometimes six-level or eight-level factors.

%MktPPro Macro

The `%MktPPro` autocall macro makes optimal partial-profile designs (Chrzan and Elrod 1995) from block designs and orthogonal arrays.* See the following pages for examples of using this macro in the discrete choice chapter: 645, 650, 654, 656, 659, and 662. Additional examples appear throughout this chapter.

An incomplete block design is a list of t treatments (in this case attributes) that appear together in b blocks. Each block contains a subset ($k < t$) of the treatments. An unbalanced block design is an incomplete block design where every treatment appears with the same frequency, but pairwise frequencies (the number of times each treatment appears with each other treatment in a block) are not constant. When both treatment and pairwise frequencies are constant, the design is a balanced incomplete block design (BIBD). We can make optimal partial-profile designs with BIBDs (the best) and with unbalanced block designs as well. While you can more generally use any incomplete block design, the results will not generally be optimal.

In a certain class of partial-profile designs, a block design specifies which attributes vary in each choice set, and an orthogonal array specifies how they vary. This next example makes an *optimal* partial-profile choice design with 16 binary attributes, four of which vary at a time in 80 choice sets. The following steps create and evaluate the partial-profile design:

```
%mktex(4 2 ** 4, n=8, seed=306)

proc sort data=randomized out=randes(drop=x1);
   by x2 x1;
   run;

proc print noobs data=randes; run;

%mktbibd(b=20, nattrs=16, setsize=4, seed=104)

%mktppro(design=randes, ibd=bibd, print=f p)

%choiceff(data=chdes,              /* candidate set of choice sets        */
          init=chdes,              /* initial design                      */
          initvars=x1-x16,         /* factors in the initial design       */
          model=class(x1-x16 / sta),/* model with stdz orthogonal coding  */
          nsets=80,                /* number of choice sets               */
          nalts=2,                 /* number of alternatives              */
          rscale=                  /* relative D-efficiency scale factor  */
          %sysevalf(80 * 4 / 16),  /* 4 of 16 attrs in 80 sets vary       */
          beta=zero)               /* assumed beta vector, Ho: b=0        */
```

The results of these steps are shown following some more explanation of the process of making a partial-profile choice design using this method. Two input designs are input to the `%MktPPro` macro. The first is an orthogonal array. The orthogonal array must be a p^k subset of an array $p^k s^1$ in $p \times s$ runs with $k \leq s$. The following designs are examples of suitable orthogonal arrays:

*The ideas implemented in this macro are based on ideas from and discussions with Don Anderson. Don is not referenced in this book nearly as much as he should be. Much of my work can be traced to conversations with Don.

2^4 in 8 runs, selected from $2^4 4^1$ in 8 runs
3^3 in 9 runs, selected from $3^3 3^1$ in 9 runs
4^4 in 16 runs, selected from $4^4 4^1$ in 16 runs
3^6 in 18 runs, selected from $3^6 6^1$ in 18 runs

You can use the `%MktOrth` or `%MktRuns` macros to see if the orthogonal array of interest exists. The following steps list the arrays that work:

```
%mktorth(options=parent, maxlev=144)

data x(keep=n design);
   set mktdeslev;
   array x[144];
   c = 0; one = 0; k = 0;
   do i = 1 to 144;
      c + (x[i] > 0); /* how many differing numbers of levels */
      if x[i] > 1 then do; p = i; k = x[i]; end; /* p^k */
      if x[i] = 1 then do; one + 1; s = i;  end; /* s^1 */
      end;
   if c = 1 then do; c = 2; one = 1; s = p; k = p - 1; end;
   if c = 2 and one = 1 and k > 2 and s * p = n;
   design = compbl(left(design));
   run;

proc print; run;
```

The rows of the orthogonal array must be sorted into the right order. Each submatrix of s rows in this kind of orthogonal array is called a difference scheme, and submatrices 2 through p are obtained from the preceding submatrix by adding 1 (in the appropriate Galois field). You will not get the right results if you use any other kind of orthogonal array.

There are several ways to create an orthogonal array that is sorted in the right way. The preceding `%MktEx` step requests the s-level factor first, then it sorts the randomized design by the first p-level factor followed by the s-level factor. Finally, it discards the s-level factor. Alternatively, you could first request one of the p-level factors, then request the s-level factor, then request the remaining $(k-1)$ p-level factors. Then in the `out=design` data set, *after* the array is created, discard `x2`, the s-level factor. Note that you cannot drop the second factor in the `%MktEx` step by specifying `out=design(drop=x2)`, because `%MktEx` will drop the variable and then sort, and your rows will be in the wrong order. The former approach has the advantage of being less likely to produce alternatives that are constant (that is, in one alternative, all attributes are absent and in other alternatives, all attributes are present).

The orthogonal array is as follows:

```
                    x2    x3    x4    x5

                     1     1     2     1
                     1     2     2     2
                     1     1     1     2
                     1     2     1     1
                     2     2     1     2
                     2     1     1     1
                     2     2     2     1
                     2     1     2     2
```

In this array, if you take the first submatrix of $s = 4$ observations and change 1 to 2 and 2 to 1, you get the second submatrix. This shows for two-level factors that the design is in the right order. More generally, when p is prime, the $i + 1$ submatrix can be made from the *ith* submatrix by adding 1 in a field with p elements. For example, with $p = 3$ and elements 1, 2, and 3, then 1 + 1 = 2, 2 + 1 = 3, 3 + 1 = 1. You do not really need to worry about any of this though, because when you create the orthogonal array using the method outlined above, your array will always be in the right order.

The second input data set contains the block design. In the example above, the BIBD is as follows:

```
                Balanced Incomplete Block Design

                    x1    x2    x3    x4

                    15    16     5     6
                     4     1    16    12
                     5    13     4     7
                     9     4     6     8
                    14     8    13    16
                    12    14     5    10
                    13     9    12     3
                     7     8     1    10
                    16     7     3    11
                     8    12    11    15
                    10     3    15     4
                     1     6     3    14
                     2    11     4    14
                    15     7    14     9
                     6     2     7    12
                    11     5     9     1
                    13    15     2     1
                    16    10     9     2
                    11     6    10    13
                     3     2     8     5
```

This design has `b=20` rows, so the final choice design will have 20 blocks of choice sets (in this case 20 blocks of 4 choice sets). The number of attributes is 6 = 16, which is specified by `nattrs=16`. The number of attributes in each block is specified by the `setsize=4` option. This design specifies that in the first block of $s = 4$ choice sets, attributes 15, 16, 5, and 6 will vary; in the second block of 4 choice sets, attributes 4, 1, 16, and 12 will vary; and so on. Alternative $i = 1$, ... p, in each block of s choice sets is made from a submatrix of s runs in the orthogonal array. For example, alternative 1 of the first s choice sets is made from the first s runs in the orthogonal array, and alternative 2 is made from the next s runs in the orthogonal array, and so on. Each attribute should appear the same number of times in the block design. The following $t = 16$ by $t = 16$ attribute frequencies matrix shows how often each attribute appears with every other attribute in our design:

```
                 Attribute by Attribute Frequencies

       1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16

 1     5   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
 2         5   1   1   1   1   1   1   1   1   1   1   1   1   1   1
 3             5   1   1   1   1   1   1   1   1   1   1   1   1   1
 4                 5   1   1   1   1   1   1   1   1   1   1   1   1
 5                     5   1   1   1   1   1   1   1   1   1   1   1
 6                         5   1   1   1   1   1   1   1   1   1   1
 7                             5   1   1   1   1   1   1   1   1   1
 8                                 5   1   1   1   1   1   1   1   1
 9                                     5   1   1   1   1   1   1   1
10                                         5   1   1   1   1   1   1
11                                             5   1   1   1   1   1
12                                                 5   1   1   1   1
13                                                     5   1   1   1
14                                                         5   1   1
15                                                             5   1
16                                                                 5
```

Each attribute appears 5 times and is paired with every other attribute exactly once. The fact that values above the diagonal are constant, in this case all ones, makes this block design a BIBD. We always want the diagonal elements to all be constant. The number of choice sets is s times the number of blocks (rows in the design), in this case, $4 \times 20 = 80$. The number of attributes is the maximum value in the BIBD (in this case, `nattrs=`$t = 16$). The number of attributes that vary at any one time is `setsize=`$k \leq s$. All factors have p levels, and all choice sets have p alternatives. The final report from the `%ChoicEff` macro is as follows:

```
                              Final Results

                        Design                   1
                        Choice Sets             80
                        Alternatives             2
                        Parameters              16
                        Maximum Parameters      80
                        D-Efficiency       20.0000
                        Relative D-Eff    100.0000
                        D-Error             0.0500
                        1 / Choice Sets     0.0125

            Variable                                      Standard
      n       Name       Label    Variance    DF           Error

      1       x11        x1 1       0.05       1         0.22361
      2       x21        x2 1       0.05       1         0.22361
      3       x31        x3 1       0.05       1         0.22361
      4       x41        x4 1       0.05       1         0.22361
      5       x51        x5 1       0.05       1         0.22361
      6       x61        x6 1       0.05       1         0.22361
      7       x71        x7 1       0.05       1         0.22361
      8       x81        x8 1       0.05       1         0.22361
      9       x91        x9 1       0.05       1         0.22361
     10       x101       x10 1      0.05       1         0.22361
     11       x111       x11 1      0.05       1         0.22361
     12       x121       x12 1      0.05       1         0.22361
     13       x131       x13 1      0.05       1         0.22361
     14       x141       x14 1      0.05       1         0.22361
     15       x151       x15 1      0.05       1         0.22361
     16       x161       x16 1      0.05       1         0.22361
                                              ==
                                              16
```

The variances are constant, and the relative *D*-efficiency is 100. *D*-efficiency is 20, which is 4 / 16 of the number of choice sets, 80. Since 25% of the attributes vary, *D*-efficiency is 25% of what we would expect in a full-profile generic choice design. We specified `rscale=20` so that relative *D*-efficiency could be based on the true maximum. The macro function `%sysevalf` is used to evaluate the expression `80 * 4 / 16` to get the 20. This function evaluates expressions that contain or produce floating point numbers and then returns the result. This particular expression could be evaluates with `%eval`, which does integer arithmetic, but that will not always be the case. In summary, this design is optimal for partial profiles, and it is 25% efficient relative to an optimal generic design where all attributes can vary.

The first three choice sets are as follows:

```
 Set  x1  x2  x3  x4  x5  x6  x7  x8  x9  x10  x11  x12  x13  x14  x15  x16

   1   1   1   1   1   2   1   1   1   1   1    1    1    1    1    1    1
       1   1   1   1   1   2   1   1   1   1    1    1    1    1    2    2

   2   1   1   1   1   2   2   1   1   1   1    1    1    1    1    1    2
       1   1   1   1   1   1   1   1   1   1    1    1    1    1    2    1

   3   1   1   1   1   1   2   1   1   1   1    1    1    1    1    1    1
       1   1   1   1   2   1   1   1   1   1    1    1    1    1    2    2
```

We have already seen that the choice design is statistically optimal. However, it might perhaps not be optimal from a practical usage point of view. In every choice set, there is a pair of alternatives, one with three attributes present and one with only one attribute present. This might not meet your needs (or maybe it does). You always need to inspect your designs to see how well they work for your purposes. Statistical efficiency and optimality are just one part of the picture. With the OA $4^1 2^4$ in 8 runs, there is one other kind of OA that you can get by specifying other random number seeds, that will always produce a choice set (one in each block of choice sets) where all four attributes are present in one alternative and none are present in the other alternative. This might be worse. This kind of problem is less likely to be an issue when you vary more than four attributes.

You can make an optimal partial-profile choice design using an unbalanced block design (instead of a BIBD). In an unbalanced block design, the treatment frequencies are constant, but the pairwise frequencies are not constant. The following example makes a design with 6 three-level factors in 60 choice sets with three alternatives each:

```
%mktex(6 3 ** 6, n=18, seed=424)

proc sort data=randomized out=randes(drop=x1);
   by x2 x1;
   run;

proc print data=randes noobs; run;

%mktbsize(nattrs=20, setsize=6, options=ubd)

%mktbibd(b=10, nattrs=20, setsize=6, seed=104)

%mktppro(design=randes, ibd=bibd, print=f p)

%choiceff(data=chdes,               /* candidate set of choice sets        */
          init=chdes,               /* initial design                      */
          initvars=x1-x20,          /* factors in the initial design       */
          model=class(x1-x20 / sta),/* model with stdz orthogonal coding   */
          nsets=60,                 /* number of choice sets               */
          nalts=3,                  /* number of alternatives              */
          rscale=                   /* relative D-efficiency scale factor  */
          %sysevalf(60 * 6 / 20),   /* 6 of 20 attrs in 60 sets vary       */
          beta=zero)                /* assumed beta vector, Ho: b=0        */
```

The `%MktBSize` macro is used to suggest the size of a block design. Each attribute appears three times in the design and each pair will occur in the range 0 to 2 times (average 0.79). The design is optimal for this specification.

The *D*-efficiency is 18 which corresponds to the 6 / 20 of 60 choice sets. Taking this into account, relative *D*-efficiency is 100. This shows that the design is optimal for partial profiles, and is 30% efficient relative to a full-profile optimal generic design where all attributes can vary.

You could also specify `b=20` in `%MktBIBD` and `nsets=120` in ChoiceEff to make larger design. Any integer multiple of `b=10` will work.

%MktPPro Macro Options

The following options can be used with the `%MktPPro` macro:

Option	Description
`help`	(positional) "help" or "?" displays syntax summary
`ibd=`*SAS-data-set*	incomplete block design
`design=`*SAS-data-set*	orthogonal array
`out=`*SAS-data-set*	output partial-profile design
`print=`*print-options*	output display options
`x=`*SAS-data-set*	incomplete block design incidence matrix

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%mktppro(help)
%mktppro(?)
```

You must specify either `ibd=` or `x=` but not both.

ibd= *SAS-data-set*

specifies the block design. Ideally, this design is a BIBD, but an unbalanced block design is fine. Also, any incomplete block design can be used.

design= *SAS-data-set*

specifies the orthogonal array from the `%MktEx` macro.

out= *SAS-data-set*

specifies the output choice design.

print= *print-options*
specifies both of the output display options. The default is `print=f`. Specify one or more values from the following list.

`i`	incomplete block design
`f`	crosstabulation of attribute frequencies
`p`	partial-profile design

x= *SAS-data-set*
specifies the incidence matrix for the block design, which is a binary coding of the design.

%MktPPro Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktRoll Macro

The `%MktRoll` autocall macro constructs a choice design from a linear arrangement. See the following pages for examples of using the `%MktRoll` macro in the design chapter: 134 and 192. Also see the following pages for examples of using this macro in the discrete choice chapter: 312, 320, 357, 387, 429, 505, 546, 556, 575, 607, 617, 628 and 636. Additional examples appear throughout this chapter. The `%MktRoll` macro takes as input a SAS data set containing an experimental design with one row per choice set, the *linear arrangement*, for example, a design created by the `%MktEx` macro. This data set is specified in the `design=` option. This data set has one variable for each attribute of each alternative in the choice experiment. The output from this macro is an `out=` SAS data set is the *choice design* containing the experimental design with one row per alternative per choice set. There is one column for each different attribute. For example, in a simple branded study, `design=` could contain the variables `x1-x5` which contain the prices of each of five alternative brands. The output data set would have one factor, `Price`, that contains the price of each of the five alternatives. In addition, it would have the number (or optionally the name) of each alternative.[*]

The rules for determining the mapping between factors in the `design=` data set and the `out=` data set are contained in the `key=` data set. For example, assume that the `design=` data set contains the variables `x1-x5` which contain the prices of each of five alternative brands: Brand A, B, C, D, and E. The choice design has two factors, `Brand` and `Price`. Brand A price is made from `x1`, Brand B price is made from `x2`, ..., and Brand E price is made from `x5`. A convenient way to get all the names in a variable list like `x1-x5` is with the `%MktKey` macro. The following step creates the five names in a single column:

```
%mktkey(5 1)
```

The `%MktKey` macro produces the following data set:

```
                                        x1

                                        x1
                                        x2
                                        x3
                                        x4
                                        x5
```

The following step creates the `Key` data set:

[*]See page 67 for an explanation of the linear arrangement of a choice design versus the arrangement of a choice design that is more suitable for analysis.

```
data key;
   input (Brand Price) ($);
   datalines;
A x1
B x2
C x3
D x4
E x5
;
```

This data set has two variables. `Brand` contains the brand names, and `Price` contains the names of the factors that are used to make the price effects for each of the alternatives. The `out=` data set will contain the variables with the same names as the variables in the `key=` data set.

The following step creates the linear arrangement with one row per choice set:

```
%mktex(3 ** 5, n=12)
```

The following step creates the choice design with one row per alternative per choice set:

```
%mktroll(design=randomized, key=key, out=sasuser.design, alt=brand)
```

Consider, for example, a randomized data set that contains the following row:

```
Obs    x1    x2    x3    x4    x5

  9     3     1     1     2     1
```

Then the data set SASUSER.DESIGN contains the following rows:

```
Obs    Set    Brand    Price

 41     9       A        3
 42     9       B        1
 43     9       C        1
 44     9       D        2
 45     9       E        1
```

The price for Brand A is made from `x1=3`, ..., and the price for Brand E is made from `x5=1`.

Now assume that there are three alternatives, each a different brand, and each composed of four factors: `Price`, `Size`, `Color`, and `Shape`. In addition, there is a constant alternative. First, the `%MktEx` macro is used to create a design with 12 factors, one for each attribute of each alternative. The following step creates the design:

```
%mktex(2 ** 12, n=16, seed=109)
```

The following step creates the `key=` data set:

```
data key;
   input (Brand Price Size Color Shape) ($); datalines;
          A     x1     x2   x3    x4
          B     x5     x6   x7    x8
          C     x9    x10  x11   x12
          None   .      .    .     .
;
```

It shows that there are three brands, A, B, and C, and also None.

Brand A is created from `Brand` = “A”, `Price` = `x1`, `Size` = `x2`, `Color` = `x3`, `Shape` = `x4`.

Brand B is created from `Brand` = “B”, `Price` = `x5`, `Size` = `x6`, `Color` = `x7`, `Shape` = `x8`.

Brand C is created from `Brand` = “C”, `Price` = `x9`, `Size` = `x10`, `Color` = `x11`, `Shape` = `x12`.

The constant alternative is created from `Brand` = “None” and none of the attributes. The “`.`” notation is used to indicate missing values in input data sets. The actual values in the `Key` data set are blank (character missing).

The following step creates the design with one row per alternative per choice set:

```
%mktroll(design=randomized, key=key, out=sasuser.design, alt=brand)
```

Consider, for example, a randomized data set that contains the following row:

```
Obs    x1    x2    x3    x4    x5    x6    x7    x8    x9   x10    x11    x12

  8     2     2     2     2     2     1     1     2     2     2      1      2
```

Then the data set SASUSER.DESIGN contains the following rows:

```
             29      8    A          2        2       2        2
             30      8    B          2        1       1        2
             31      8    C          2        2       1        2
             32      8    None       .        .       .        .
```

Now assume like before that there are three branded alternatives, each composed of four factors: `Price`, `Size`, `Color`, and `Shape`. In addition, there is a constant alternative. Also, there is an alternative-specific factor, `Pattern`, that only applies to Brand A and Brand C. First, the `%MktEx` macro is used to create a design with 14 factors, one for each attribute of each alternative. The following step creates the design:

```
%mktex(2 ** 14, n=16, seed=114)
```

The following step creates the `key=` data set:

```
data key;
   input (Brand Price Size Color Shape Pattern) ($);
   datalines;
A     x1     x2   x3    x4   x13
B     x5     x6   x7    x8     .
C     x9    x10  x11   x12   x14
None   .      .    .     .     .
;
```

It shows that there are three brands, A, B, and C, plus None.

Brand A is created from `Brand` = "A", `Price` = `x1`, `Size` = `x2`, `Color` = `x3`, `Shape` = `x4`, `Pattern` = `x13`.

Brand B is created from `Brand` = "B", `Price` = `x5`, `Size` = `x6`, `Color` = `x7`, `Shape` = `x8`.

Brand C is created from `Brand` = "C", `Price` = `x9`, `Size` = `x10`, `Color` = `x11`, `Shape` = `x12`, `Pattern` = `x14`.

The constant alternative is `Brand` = "None" and none of the attributes.

The following step creates the design with one row per alternative per choice set:

```
%mktroll(design=randomized, key=key, out=sasuser.design, alt=brand)
```

Consider, for example, a randomized data set that contains the following row:

```
 Obs   x1   x2   x3   x4   x5   x6   x7   x8   x9   x10   x11   x12   x13   x14

   8    2    1    1    2    1    2    1    2    1    1     1     2     1     2
```

Then the data set SASUSER.DESIGN contains the following rows:

```
Obs    Set    Brand    Price    Size    Color    Shape    Pattern

 29     8     A          2        1       1        2          1
 30     8     B          1        2       1        2          .
 31     8     C          1        1       1        2          2
 32     8     None       .        .       .        .          .
```

Now assume we are going to fit a model with price cross-effects so we need `x1`, `x5`, and `x9` (the three price effects) available in the `out=` data set. See pages 444 and 468 for other examples of cross-effects. The following step creates the design:

```
%mktroll(design=randomized, key=key, out=sasuser.design, alt=brand,
         keep=x1 x5 x9)
```

Now the data set also contains the three original price variables, for example, as follows:

```
Obs   Set   Brand   Price   Size   Color   Shape   Pattern   x1   x5   x9

 29     8   A         2       1      1       2        1       2    1    1
 30     8   B         1       2      1       2        .       2    1    1
 31     8   C         1       1      1       2        2       2    1    1
 32     8   None      .       .      .       .        .       2    1    1
```

Every value in the `key=` data set must appear as a variable in the `design=` data set. The macro displays a warning if it encounters a variable name in the `design=` data set that does not appear as a value in the `key=` data set.

%MktRoll Macro Options

The following options can be used with the `%MktRoll` macro:

Option	Description
`help`	(positional) "help" or "?" displays syntax summary
`alt=`*variable*	variable with name of each alternative
`design=`*SAS-data-set*	input SAS data set
`keep=`*variable-list*	factors to keep
`key=`*SAS-data-set*	`Key` data set name
`key=`*rows columns* <`t`>	`Key` data set description
`options=nowarn`	suppress the variables not mentioned warning
`out=`*SAS-data-set*	output SAS data set
`set=`*variable*	choice set number variable

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%mktroll(help)
%mktroll(?)
```

You must specify the `design=` and `key=` options.

alt= *variable*

specifies the variable in the `key=` data set that contains the name of each alternative. Often this is something like `alt=Brand`. When `alt=` is not specified, the macro creates a variable `_Alt_` that contains the alternative number.

design= *SAS-data-set*

specifies an input SAS data set with one row per choice set. The `design=` option must be specified.

keep= *variable-list*
specifies factors from the `design=` data set that should also be kept in the `out=` data set. This option is useful to keep terms that are used to create cross-effects.

key= *SAS-data-set* | *l*ist
specifies the rules for mapping the `design=` data set to the `out=` data set. The `key=` option must be specified. It has one of two forms. 1) The `key=` option names an input SAS data set containing the rules for mapping the `design=` data set to the `out=` data set. The structure of this data set is described in detail in the preceding examples. 2) When you want the `key=` data set to exactly match the data set that comes out of the MktKey macro, you can specify the argument to the MktKey macro directly in the `key=` option, and the `%MktRoll` macro will make the `key=key` data set for you. In other words, the following two specifications are equivalent:

```
%mktkey(3 3 t)
%mktroll(design=design, key=key,   out=frac)

%mktroll(design=design, key=3 3 t, out=frac)
```

options= *options-list*
specifies binary options. By default, none of these options are specified. Specify one or more of the following values after `options=`.

`nowarn`
does not display a warning when the `design=` data set contains variables not mentioned in the `key=` data set. Sometimes this is perfectly fine.

out= *SAS-data-set*
specifies the output SAS data set. If `out=` is not specified, the DATAn convention is used.

set= *variable*
specifies the variable in the `out=` data set that will contain the choice set number. By default, this variable is named `Set`.

%MktRoll Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktRuns Macro

The `%MktRuns` autocall macro suggests reasonable sizes for experimental designs. See the following pages for examples of using this macro in the design chapter: 128, 188 and 199. Also see the following pages for examples of using this macro in the discrete choice chapter: 302, 340, 411, 415, 482, 483, 535 and 557. Additional examples appear throughout this chapter. The `%MktRuns` macro tries to find sizes in which perfect balance and orthogonality can occur, or at least sizes in which violations of orthogonality and balance are minimized. Typically, the macro takes one argument, a list of the number of levels of each factor.

For example, with 3 two-level and 4 three-level factors, you can specify either of the following:

```
%mktruns(2 2 2 3 3 3 3)

%mktruns(2 ** 3 3 ** 4)
```

The output from the macro is as follows:

```
                                Design Summary

                            Number of
                            Levels       Frequency

                                2            3
                                3            4

                    Saturated      = 12
                    Full Factorial = 648

                    Some Reasonable                      Cannot Be
                       Design Sizes       Violations     Divided By

                                36 *               0
                                72 *               0
                                18                 3     4
                                54                 3     4
                                12 S               6     9
                                24                 6     9
                                48                 6     9
                                60                 6     9
                                30                 9     4 9
                                42                 9     4 9

              * - 100% Efficient design can be made with the MktEx macro.
              S - Saturated Design - The smallest design that can be made.
```

```
  n    Design                                            Reference

 36    2 ** 16  3 **  4                               Orthogonal Array
 36    2 ** 11  3 ** 12                               Orthogonal Array
 36    2 ** 10  3 **  8   6 **  1                     Orthogonal Array
 36    2 **  9  3 **  4   6 **  2                     Orthogonal Array
 36    2 **  4  3 ** 13                               Orthogonal Array
 36    2 **  3  3 **  9   6 **  1                     Orthogonal Array
 72    2 ** 52  3 **  4                               Orthogonal Array
 72    2 ** 49  3 **  4   4 **  1                     Orthogonal Array
 72    2 ** 47  3 ** 12                               Orthogonal Array
 72    2 ** 46  3 **  8   6 **  1                     Orthogonal Array
 72    2 ** 45  3 **  4   6 **  2                     Orthogonal Array
  .
  .
  .
```

The macro reports that the saturated design has 12 runs and that 36 and 72 are optimal design sizes. The macro picks 36, because it is the smallest integer >= 12 that can be divided by 2, 3, 2×2, 2×3, and 3×3. The macro also reports 18 as a reasonable size. There are three violations with 18, because 18 cannot be divided by each of the three pairs of 2×2, so perfect orthogonality in the two-level factors will not be possible with 18 runs. Larger sizes are reported as well. The macro displays orthogonal designs that are available from the `%MktEx` macro that match your specification.

You can run PROC PRINT after the macro finishes to see every size the macro considered, for example, as follows:

```
proc print label data=nums split='-';
   label s = '00'x;
   id n;
   run;
```

The output from this step is not shown.

For 2 two-level factors, 2 three-level factors, 2 four-level factors, and 2 five-level factors specify the following:

```
%mktruns(2 2 3 3 4 4 5 5)
```

The results are as follows:

```
                              Design Summary

                          Number of
                          Levels       Frequency

                               2           2
                               3           2
                               4           2
                               5           2

    Saturated      = 21
    Full Factorial = 14,400

    Some Reasonable                         Cannot Be
       Design Sizes       Violations      Divided By

                120                3      9 16 25
                180                6      8 16 25
                 60                7      8  9 16 25
                144               15      5 10 15 20 25
                 48               16      5  9 10 15 20 25
                 72               16      5 10 15 16 20 25
                 80               16      3  6  9 12 15 25
                 96               16      5  9 10 15 20 25
                160               16      3  6  9 12 15 25
                192               16      5  9 10 15 20 25
                 21 S             34      2  4  5  6  8  9 10 12 15 16 20 25

           S - Saturated Design - The smallest design that can be made.
               Note that the saturated design is not one of the
               recommended designs for this problem.  It is shown
               to provide some context for the recommended sizes.
```

Among the smaller design sizes, 60 or 48 look like good possibilities.

The macro has an optional keyword parameter: `max=`. It specifies the maximum number of sizes to try. Usually you will not need to specify the `max=` option. The smallest design that is considered is the saturated design. The following specification tries 5000 sizes (21 to 5020) and reports that a perfect design can be found with 3600 runs:

```
%mktruns(2 2 3 3 4 4 5 5, max=5000)
```

The results are as follows:

```
                                  Design Summary

                              Number of
                              Levels       Frequency

                                   2           2
                                   3           2
                                   4           2
                                   5           2

                    Saturated      = 21
                    Full Factorial = 14,400

       Some Reasonable                       Cannot Be
          Design Sizes       Violations      Divided By

                  3600                0
                   720                1      25
                  1200                1       9
                  1440                1      25
                  1800                1      16
                  2160                1      25
                  2400                1       9
                  2880                1      25
                  4320                1      25
                  4800                1       9
                    21 S             34       2   4   5   6   8   9  10  12  15  16  20  25

              S - Saturated Design - The smallest design that can be made.
                  Note that the saturated design is not one of the
                  recommended designs for this problem.  It is shown
                  to provide some context for the recommended sizes.
```

The `%MktEx` macro does not explicitly know how to make this design, however, it can usually find it or come extremely close with the coordinate exchange algorithm.

Now consider again the problem with 3 two-level and 4 three-level factors, but this time we want to estimate the interaction of two of the two-level factors. The following step runs the macro:

```
   %mktruns(2 2 2 3 3 3 3, interact=1*2, options=source)
```

Since `options=source` was specified, the first part of the following output lists the sources for orthogonality violations that the macro will consider. We see that n must be divided by: 2 since `x1-x3` are two-level factors, 3 since `x4-x7` are three-level factors, 4 since `x1*x2`, `x1*x3`, and `x2*x3` interactions are specified, 6 since we have two-level and three-level factors, 8 since we have the two-way interaction of 2 two-level factors and the main-effect of an additional two-level factor, 9 since we have multiple three-level factors, and 12 since we have the two-way interaction of 2 two-level factors and the main-effect of additional three-level factors. The results are as follows:

```
                                       Failing to
                                       Divide by
                                       the Divisor
                                       Contributes
                                       this to the
                                       Violations
To Achieve Both    The Source of       Count
Orthogonality      the Divisor                          The Divisor
and Balance,                           The Number of    Comes
N Must Be          Number of           Sources of       from these
Divided By         Levels or Cells     the Divisor      Factors

        2               2                    3             1
                                                           2
                                                           3

        3               3                    4             4
                                                           5
                                                           6
                                                           7

        4               2*2                  3             1 2
                                                           1 3
                                                           2 3

        6               2*3                 12             1 4
                                                           1 5
                                                           1 6
                                                           1 7
                                                           2 4
                                                           2 5
                                                           2 6
                                                           2 7
                                                           3 4
                                                           3 5
                                                           3 6
                                                           3 7

        8               2*2*2                1             1 2 3

        9               3*3                  6             4 5
                                                           4 6
                                                           4 7
                                                           5 6
                                                           5 7
                                                           6 7
```

```
      12              2*2*3                  4             1 2 4
                                                           1 2 5
                                                           1 2 6
                                                           1 2 7


                               Design Summary

                           Number of
                           Levels       Frequency

                                2           3
                                3           4

             Saturated      = 13
             Full Factorial = 648

             Some Reasonable                      Cannot Be
                Design Sizes       Violations     Divided By

                         72                 0
                        144                 0
                         36                 1     8
                        108                 1     8
                         48                 6     9
                         96                 6     9
                        120                 6     9
                         60                 7     8  9
                         84                 7     8  9
                         13 S              33     2  3  4  6  8  9 12

            S - Saturated Design - The smallest design that can be made.
                Note that the saturated design is not one of the
                recommended designs for this problem.  It is shown
                to provide some context for the recommended sizes.
```

Now we need 72 runs for perfect balance and orthogonality although the `%MktEx` design catalog is not guaranteed to contain designs with interactions.

%MktRuns Macro Options

The following options can be used with the `%MktRuns` macro:

Option	Description
`list`	(positional) numbers of levels of all the factors
	(positional) "help" or "?" displays syntax summary
`interact=`*interaction-list*	interaction terms
`max=`*n* $< m >$ $<$ f $>$	largest design sizes to try

Option	Description
`n=`*n*	design size to evaluate
`maxlev=`*n*	maximum number of levels
`maxoa=`*n*	maximum number of orthogonal arrays
`options=justparse`	used by other `Mkt` macros to parse the list
`options=multiple`	allow terms to be counted multiple times
`options=multiple2`	`option=multiple` and more detailed output
`options=noprint`	suppress the display of all output
`options=nosat`	suppress the saturated design from the design list
`options=source`	displays source of numbers in design sizes
`options=512`	adds some designs in 512 runs
`out=`*SAS-data-set*	data set with the suggested sizes
`outorth=`*SAS-data-set*	data set with orthogonal array list
`toobig=`*n*	specifies problem that is too big

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%mktruns(help)
%mktruns(?)
```

The `%MktRuns` macro has one positional parameter, `list`, and several keyword parameters.

list

specifies a list of the numbers of levels of all the factors. For example, for 3 two-level factors specify either `2 2 2` or `2 ** 3`. Lists of numbers, like `2 2 3 3 4 4` or a *levels**number of factors* syntax like: `2**2 3**2 4**2` can be used, or both can be combined: `2 2 3**4 5 6`. The specification `3**4` means 4 three-level factors. You must specify a list. Note that the factor list is a positional parameter. This means it must come first, and unlike all other parameters, it is not specified after a name and an equal sign.

interact= *interaction-list*

specifies interactions that must be estimable. By default, no interactions are guaranteed to be estimable. Examples:
`interact=x1*x2`
`interact=x1*x2 x3*x4*x5`
`interact=x1|x2|x3|x4|x5@2`

The interaction syntax is in most ways like PROC GLM's and many of the other modeling procedures. It uses "`*`" for simple interactions (`x1*x2` is the interaction between `x1` and `x2`), "`|`" for main effects and interactions (`x1|x2|x3` is the same as `x1 x2 x1*x2 x3 x1*x3 x2*x3 x1*x2*x3`) and "`@`" to eliminate higher-order interactions (`x1|x2|x3@2` eliminates `x1*x2*x3` and is the same as `x1 x2 x1*x2 x3 x1*x3 x2*x3`). The specification "`@2`" creates main effects and two-way interactions. Unlike PROC GLM's syntax, some short cuts are permitted. For the factor names, you can specify either the actual variable names (for example, `x1*x2` ...) or you can just specify the factor number without the "`x`" (for example, `1*2`). You can also specify `interact=@2` for all main effects and two-way interactions omitting the `1|2|...`. The following three specifications are equivalent:

```
%mktruns(2 ** 5, interact=@2)
%mktruns(2 ** 5, interact=1|2|3|4|5@2)
%mktruns(2 ** 5, interact=x1|x2|x3|x4|x5@2)
```

max= *n* < *m* > < f >

specifies the maximum number of design sizes to try. By default, `max=200 2 f`. The macro tries up to *n* sizes starting with the saturated design. The macro stops trying larger sizes when it finds a design size with zero violations that is *m* times as big as a previously found size with zero violations. The macro reports the best 10 sizes. For example, if the saturated design has 10 runs, and there are zero violations in 16 runs, then by default, the largest size that the macro will consider is $32 = 2 \times 16$ runs. A third optional value of 'f' or 'F' is specified by default, which instructs the `%MktRuns` macro to not consider designs larger than full-factorials. If you specify `max=` without this option, larger designs might be considered.

maxlev= *n*

specifies the maximum number of levels to consider when generating the orthogonal array list. The default is `maxlev=50`, and the actual maximum is the max of the specified `maxlev=` value and the maximum number of levels in the factor list. Specify a value $2 \le n \le 144$.

n= *n*

specifies the design size to evaluate. By default, this option is not specified, and the `max=` option specification provides a range of design sizes to evaluate.

maxoa= *n*

specifies the maximum number of orthogonal arrays to display in the list of orthogonal arrays that the `%MktEx` macro knows how to make. By default, when no value is specified, the entire list is displayed. You can specify, for example, `maxoa=0, outorth=oa`, to get a data set instead of displaying this table. You could instead specify `maxoa=10` to just see the first ten arrays.

options= *options-list*

specifies binary options. By default, none of these options are specified. Specify one the following values after `options=`.

`justparse`
is used by other `Mkt` macros to have this macro just parse the list argument and return it as a simple list of integers.

`multiple`
specifies that a term that is required for orthogonality can be counted multiple times when counting orthogonality violations. For example, combinations of levels for the pair of variable 1 and variable 2 must have equal frequencies for orthogonality in the main effects, and if two-way interactions are required as well, then the (1, 2) pair must have equal frequencies again. By default, each combination of variables is only counted once. The difference between the single and multiple sources is the single source method just counts the places in the design where equal frequencies must occur. The multiple source method weights this count by the number of times each of these terms is important for achieving orthogonality. The results for the two methods are often highly correlated, but they can be different.

`multiple2`
specifies both `option=multiple` and more detailed output including the reason each term appears is added to the source table when `options=source multiple2` is specified.

`noprint`
suppresses the display of all output. The only output from the macro is the output data sets.

`nosat`
suppresses inclusion of the saturated design in the design list. By default, the saturated design is reported along with the reasonable design sizes, even if it is not a very reasonable size, in order to provide some context for the other numbers. If you specify `options=nosat`, then the saturated design will only be added to the list if it meets the usual criteria for being a reasonable size.

`source`
displays the source of all of the numbers in the table of reasonable design sizes.

`512`
adds some larger designs in 512 runs with mixes of 16, 8, 4, and 2-level factors to the catalog, which gives added flexibility in 512 runs at a cost of much slower run time. This option replaces the default $4^{160}32^1$ parent with $16^{32}32^1$.

out= *SAS-data-set*

specifies the name of a SAS data set with the suggested sizes. The default is `out=nums`.

outorth= *SAS-data-set*

specifies output data set with orthogonal array list. By default, this data set is not created.

toobig= *n*

is used to flag problems that are too big and take too long before they consume large quantities of resources. The default is `toobig=11400`. The number is the maximum number of levels and pairs of levels that is considered. With models with interactions, the interaction terms contribute to this number as well. A main-effects model with 150 factors is small enough ($150 \times 151/2 = 11325$) to work with the default `toobig=11400`. If you want to use the `%MktRuns` macro with larger models, you will have to specify a larger value for `toobig=`. The following step works with `toobig=` specified, but it is too large to work without it being specified:

```
%mktruns(2 ** 17, interact=@2, toobig=20000)
```

%MktRuns Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%Paint Macro

The `%Paint` autocall macro interpolates between colors. This macro is not a part of the experimental design or marketing research family of macros. It exists for totally different reasons. This macro has two main uses.

- You can use it with ODS. The macro can create a macro that specifies an `ods cellstyle as` statement for use in PROC TEMPLATE for coloring the background around entries in a table.
- You can use it with PROC INSIGHT. The macro can read an input `data=` data set and create an `out=` data set with a new variable `_obstat_` that contains observation symbols and colors interpolated from the `colors=` list based on the values `var=` variable. Three interpolation methods are available with the `level=` option.

This macro creates temporary work data sets named `tempdata`, `tempdat2`, `tempdat3`, and `tempdat4`.

It makes a temporary macro called "_".

When a `cellstyle` macro is not created, the paint macro by default creates an `out=` data set named `colors`.

%Paint Macro Options

The following options can be used with the `%Paint` macro:

Option	Description
`help`	(positional) "help" or "?" displays syntax summary
`values=`*do-list*	input data values
`var=`*variable-name*	input variable
`data=`*SAS-data-set*	input SAS data set
`out=`*SAS-data-set*	output SAS data set
`macro=`*macro-name*	SAS macro name
`colors=`<*color-list*> <*data-value-list*>	colors list
`level=`*measurement-level*	measurement level of the data
`symbols=`*symbols-list*	list of symbols
`format=`*SAS-format*	nominal and ordinal variable format
`order=`*summary-order*	nominal and ordinal variable order
`select=`*n*	selection state
`show=`*n*	show/hide state
`include=`*n*	include/exclude state
`label=`*n*	label/unlabel state
`rgbround=`*rounding-list*	RGB value rounding instructions
`missing=`*missing-specification*	missing value handling
`debug=`*debug-string*	debugging information to display

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%paint(help)
%paint(?)
```

One of the following two options must be specified:

values= *do-list*

specifies the input data values. This option primarily exists for use with ODS. You can tell the macro to create input data set based on a `do` list provided with this option. Example: `values=0 to 10 by 0.25`.

var= *variable-name*

specifies a variable with the input data values. This option primarily exists for use with PROC INSIGHT. Use this option when there is an input `data=` data set.

All of these next options can optionally be specified:

data= *SAS-data-set*

specifies the input SAS data set (when `var=` but not `values=` is specified). This option primarily exists for use with PROC INSIGHT.

out= *SAS-data-set*

specifies the output SAS data set. The default, when `macro=` is not specified, is `out=colors`. This option primarily exists for use with PROC INSIGHT.

macro= *macro-name*

specifies the name of the SAS macro to create instead of creating an `out=` data set. This option primarily exists for use with ODS. By default, no macro is created.

colors= *<color-list> <data-value-list>*

specifies the colors list. The colors must be selected from: red, green, blue, yellow, magenta, cyan, black, white, orange, brown, gray, olive, pink, purple, violet. For other colors, specify the RGB color name (CX*rrggbb* where *rr* is the red value, *gg* is the green, and *bb* is blue, all three specified in hex, 00 to FF). When no list is specified, the default is `colors=blue magenta red`. The option `colors=red green 1 10`, interpolates between red and green, based on the values of the `var=` variable, where values of 1 or less map to red, values of 10 or more map to green, and values in between map to colors in between. The option `colors=red yellow green 1 5 10`, interpolates between red at 1, yellow at 5, and green at 10. If the data value list is omitted it is computed from the data.

level= *measurement-level*

specifies the measurement level of the data. Values include `interval`, `ordinal`, and `nominal`. The option `level=interval` (the default) interpolates on the actual values. The option `level=ordinal` interpolates on the ranks of the input values. This can even work with character variables; the ranks are just the category numbers. The option `level=nominal` specifies that there is one color or symbol per category. Only the first three characters of the measurement level are checked.

symbols= *symbols-list*

provides a list of symbols from page 323 of the Version 6 PROC INSIGHT manual. Specify integers from 1 to 8 or select from: square plus circle diamond x up down star. Note that 'triangle' is not specified with 'up' and 'down'. Equivalent examples: `symbols=1 2 3 4 5 6`, `symbols=square plus circle diamond x up`, `symbols=1 plus circle 4 5 up`. Note that when `level=interval`, only the first symbol is used. With `level=nominal` and `level=ordinal`, the first symbol is used for the first category, the second symbol is used for the second category, ..., and if there are more categories than symbols, the last symbol is reused for all subsequent categories. Extra symbols are ignored. The default first symbol is circle, and the last symbol is substituted for invalid symbols.

format= *SAS-format*

specifies the format for the values used with `level=nominal` and `level=ordinal` variables.

order= *summary-order-option*

specifies the PROC SUMMARY `order=` option for `level=ordinal` and `level=nominal`.

These next four options specify the first four columns of `_obstat_` variable from page 323 of the Version 6 PROC INSIGHT manual.

Column 5, the symbol, is specified with `symbols=`. The values must be numeric expressions including constants, variables, and arithmetic expressions. All nonzero expression results are converted to 1.

select= *n*

specifies the selection state. The value 0 means not selected. This option primarily exists for use with PROC INSIGHT.

show= *n*

specifies the show/hide state. The value 0 means hide. This option primarily exists for use with PROC INSIGHT.

include= *n*

specifies the include/exclude state. The value 0 means exclude. This option primarily exists for use with PROC INSIGHT.

label= *n*

specifies the label/unlabel state. The value 0 means unlabel. This option primarily exists for use with PROC INSIGHT.

rgbround= *rounding-list*

specifies instructions for rounding the RGB values. The first value is used to round the `colors=var` variable. Specify a positive value to have the variable rounded to multiples of that value. Specify a negative value n to have a maximum of abs(n) colors. For the other three values, specify positive values. The last three values are rounding factors for the red, green, and blue component of the color. By default, when a value is missing, there is no rounding. The default is `rgbround=-99 1 1 1`, which creates a maximum of 99 colors and rounds the RGB values to integers. You could specify larger values for the last three values to have, for examples, the color values be multiples of two or four.

missing=

specifies how missing values in the `var=` variable are handled. By default, when `missing=` is null, the observation is not shown (`show=0` is set). The specified value is a color followed by a symbol. When only one value is specified in the `symbols=` list, the symbol is optional, and the `symbols=` symbol is used.

debug= *debug-string*

specifies types of debugging information to display. The option `debug=vars` displays macro options and macro variables for debugging. The option `debug=dprint` displays intermediate data sets. The option `debug=notes` suppresses the specification of `options nonotes` during most of the macro. The option `debug=time` displays the total macro run time. The option `debug=mprint` runs the macro with `options mprint`. Create a list for more than one type of debugging. Example: `debug=vars dprint notes time mprint`.

%PHChoice Macro

The `%PHChoice` autocall macro customizes the output from PROC PHREG for choice modeling. Typically, you run the following macro once to customize the PROC PHREG output as follows:

```
%phchoice(on)
```

The macro uses PROC TEMPLATE and ODS (Output Delivery System) to customize the output from PROC PHREG. Running this code edits the templates and stores copies in SASUSER. These changes will remain in effect until you delete them. Note that these changes assume that each effect in the choice model has a variable label associated with it so there is no need to display variable names. If you are coding with PROC TRANSREG, this will usually be the case. You can submit the following step to return to the default output from PROC PHREG:

```
%phchoice(off)
```

If you ever have errors running this macro, like invalid page errors, see "Macro Errors" on page 1211. The rest of this section discusses the details of what the `%PHChoice` macro does and why. Unless you are interested in further customization of the output, you should skip to "`%PHChoice` Macro Options" on page 1177.

We are most interested in the `Analysis of Maximum Likelihood Estimates` table, which contains the parameter estimates. We can first use PROC TEMPLATE to identify the template for the parameter estimates table and then edit the template. First, let's have PROC TEMPLATE display the templates for PROC PHREG. The `source stat.phreg` statement in the following step specifies that we want to see PROC TEMPLATE source code for the STAT product and the PHREG procedure:

```
proc template;
   source stat.phreg;
   run;
```

If we search the results for the `Analysis of Maximum Likelihood Estimates` table we find the following code, which defines the `Stat.Phreg.ParameterEstimates` table:

```
define table Stat.Phreg.ParameterEstimates;
   notes "Parameter Estimates Table";
   dynamic Confidence NRows;
   column Variable DF Estimate StdErr StdErrRatio ChiSq ProbChiSq HazardRatio
      HRLowerCL HRUpperCL Label;
   header h1 h2;

   define h1;
      text "Analysis of Maximum Likelihood Estimates";
      space = 1;
      spill_margin;
   end;
```

```
define h2;
   text Confidence BEST8. %nrstr("%% Hazard Ratio Confidence Limits");
   space = 0;
   end = HRUpperCL;
   start = HRLowerCL;
   spill_margin = OFF;
end;

define Variable;
   header = "Variable";
   style = RowHeader;
   id;
end;

define DF;
   parent = Common.ParameterEstimates.DF;
end;

define Estimate;
   header = ";Parameter;Estimate;";
   format = D10.;
   parent = Common.ParameterEstimates.Estimate;
end;

define StdErr;
   header = ";Standard;Error;";
   format = D10.;
   parent = Common.ParameterEstimates.StdErr;
end;

define StdErrRatio;
   header = ";StdErr;Ratio;";
   format = 6.3;
end;

define ChiSq;
   parent = Stat.Phreg.ChiSq;
end;

define ProbChiSq;
   parent = Stat.Phreg.ProbChiSq;
end;

define HazardRatio;
   header = ";Hazard;Ratio;";
   glue = 2;
   format = 8.3;
end;

define HRLowerCL;
   glue = 2;
   format = 8.3;
   print_headers = OFF;
end;
```

```
      define HRUpperCL;
         format = 8.3;
         print_headers = OFF;
      end;

      define Label;
         header = "Variable Label";
      end;

      col_space_max = 4;
      col_space_min = 1;
      required_space = NRows;
   end;
```

It contains header, format, spacing and other information for each column in the table. Most of this need not concern us now.[*] The template contains the following `column` statement, which lists the columns of the table:

```
      column Variable DF Estimate StdErr StdErrRatio ChiSq ProbChiSq HazardRatio
         HRLowerCL HRUpperCL Label;
```

Since we will usually have a label that adequately names each parameter, we do not need the variable column. We also do not need the hazard information. If we move the label to the front of the list and drop the variable column and the hazard columns, we get the following:

```
   column Label DF Estimate StdErr ChiSq ProbChiSq;
```

We use the `edit` statement to edit the template. We can also modify some headers. We specify the new `column` statement and the new headers. We can also modify the Summary table, which is `Stat.Phreg.CensoredSummary`, to use the vocabulary of choice models instead of survival analysis models. The code is generated by the PROC TEMPLATE step with the `source` statement. The overall header "Summary of the Number of Event and Censored Values" is changed to "Summary of Subjects, Sets, and Chosen and Unchosen Alternatives", "Total" is changed to "Number of Alternatives", "Event" is changed to "Chosen Alternatives", "Censored" is changed to "Not Chosen", and "Percent Censored" is dropped. Finally `Style=RowHeader` was specified on the label column. This sets the color, font, and general style for HTML output. The `RowHeader` style is typically used on first columns that provide names or labels for the rows. The following step contains the code that the `%phchoice(on)` macro runs:

```
   proc template;
      edit stat.phreg.ParameterEstimates;
         column Label DF Estimate StdErr ChiSq ProbChiSq;
         header h1;

         define h1;
            text "Multinomial Logit Parameter Estimates";
            space = 1;
            spill_margin;
            end;
```

[*]In fact, this is an older version of the template. It has changed. Still, this is the version that the macro was based on, so it is still shown here.

```
         define Label;
            header = " " style = RowHeader;
            end;
         end;

         edit Stat.Phreg.CensoredSummary;
            column Stratum Pattern Freq GenericStrVar Total
                   Event Censored;
            header h1;
            define h1;
               text "Summary of Subjects, Sets, "
                    "and Chosen and Unchosen Alternatives";
               space = 1;
               spill_margin;
               first_panel;
            end;

            define Freq;
              header=";Number of;Choices" format=6.0;
            end;

         define Total;
            header = ";Number of;Alternatives";
            format_ndec = ndec;
            format_width = 8;
         end;

         define Event;
            header = ";Chosen;Alternatives";
            format_ndec = ndec;
            format_width = 8;
         end;

         define Censored;
            header = "Not Chosen";
            format_ndec = ndec;
            format_width = 8;
         end;
         end;

      run;
```

The `%phchoice(off)` macro runs the following step:

```
   * Delete edited templates, restore original templates;
   proc template;
      delete Stat.Phreg.ParameterEstimates;
      delete Stat.Phreg.CensoredSummary;
      run;
```

Our editing of the multinomial logit parameter estimates table assumes that each independent variable has a label. If you are coding with PROC TRANSREG, this is true of all variables created by `class` expansions. You might have to provide labels for `identity` and other variables. Alternatively, if you want variable names to appear in the table, you can do that as follows:

```
    %phchoice(on, Variable DF Estimate StdErr ChiSq ProbChiSq Label)
```

This might be useful when you are not coding with PROC TRANSREG. The optional second argument provides a list of the column names to display. The available columns are: `Variable DF Estimate StdErr StdErrRatio ChiSq ProbChiSq HazardRatio HRLowerCL HRUpperCL Label`. (`HRLowerCL` and `HRUpperCL` are confidence limits on the hazard ratio.) For very detailed customizations, you might have to run PROC TEMPLATE directly.

%PHChoice Macro Options

The following options can be used with the `%PHChoice` macro:

Option	Description
`onoff`	(positional) `on`, `off`, or `expb`
	(positional) “help” or “?” displays syntax summary
`column`	(positional) list of columns

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
    %phchoice(help)
    %phchoice(?)
```

The `%PHChoice` macro has two positional parameters, `onoff` and `column`. Positional parameters must come first, and unlike all other parameters, are not specified after a name and an equal sign.

onoff

`ON` specifies choice model customization.
`OFF` turns off the choice model customization and returns to the default PROC PHREG templates.
`EXPB` turns on choice model customization and adds the hazard ratio to the output.
Upper/lower case does not matter.

column

specifies an optional column list for more extensive customizations.

%PlotIt Macro

The %PlotIt autocall macro makes graphical scatter plots of labeled points. It is particularly designed to display raw data and results from analyses such as regression, correspondence analysis, MDPREF, PREFMAP, and MDS. However, it can make many other types of graphical displays as well. It can plot points, labeled points, vectors, circles and density. See pages 27–40 and 1231–1274, for example, plots and more about these methods. The %PlotIt macro is not needed nearly as much now as it was in previous SAS releases. Now, ODS Graphics automatically does much of what the %PlotIt macro was originally designed for, and usually, ODS Graphics does it better and more conveniently. Comparisons between the %PlotIt macro and ODS Graphics are as follows. The %PlotIt macro has a much more sophisticated algorithm for placing labels and avoiding label collisions. The %PlotIt macro has a few other capabilities that are not in ODS Graphics (e.g. more sophisticated color ramps or "painting" for some applications). ODS Graphics is superior to the %PlotIt macro in virtually every other way, and ODS Graphics has *many* capabilities that the %PlotIt macro does not have. The algorithm that ODS Graphics has for label placement in 9.2, while clearly nonoptimal, is good enough for many analyses.

The macro, by default, uses the last data set created. The macro creates an output Annotate data set that cannot be used as input to the macro, so you must specify data= if you run %PlotIt a second time.

By default (at least by default when a device like WIN is in effect on a PC), the %PlotIt macro creates a graphical scatter plot on your screen. If no graphics device has been previously specified (either directly or indirectly), you will be prompted for a device as follows:

```
No device name has been given--please enter device name:
```

Enter your graphics device. This name is remembered for the duration of your SAS session or until you change the device. You can modify the gopprint= and gopplot= options to set default devices so that you will not be prompted. Note that all graphics options specified in a goptions statement (except device=) are ignored by default. Use the macro options gopprint=, gopplot=, gopts2=, and gopts= to set goptions.

To display a plot on your screen using the default goptions in a PC environment, specify, for example, the following:

```
goptions device=win;
%plotit(data=sashelp.class(drop=age sex))
```

To create a PDF file named myplot.pdf, suitable for printing, specify the following:

```
ods listing style=statistical;
%plotit(data=sashelp.class(drop=age sex),
        gopts=device=pdf, method=print, post=myplot.pdf)
```

The ODS LISTING statement is only necessary in this example if you want to set the ODS style.

To create HTML output, specify the following:

```
goptions device=png;
ods listing close;
ods html body='b.html';
%plotit(data=sashelp.class(drop=age sex))
ods html close;
```

To create RTF output, specify the following:

```
goptions device=png;
ods listing close;
ods rtf file='myplot.rtf';
%plotit(data=sashelp.class(drop=age sex))
ods rtf close;
```

To create a PNG file, perhaps just to insert into a document, specify the following:

```
goptions device=png;
ods listing style=analysis;
%plotit(data=sashelp.class(drop=age sex), method=print, post=myplot.png)
ods listing close;
```

Use `gout=` to write the plot to a catalog.

The default color scheme and a few other options have changed with this release in ways consistent with SAS/GRAPH procedures. Like SAS/GRAPH procedures in 9.2, the appearance of the output is in part controlled by the `gstyle` system option. The following statements enable and disable this option:

```
options gstyle;
options nogstyle;
```

With `gstyle` in effect (which will typically be the default), the output will look much different (and probably better) than it looked in previous releases. In part, this is due to the defaults for several options changing. For example, with `gstyle`, the macro will use hardware fonts by default with the style instead of Swiss software fonts. Mostly, however, the differences are due to the output being at least in part controlled by the ODS style now. Specify the `nogstyle` system option to get the old appearance. Note that by default, the listing ODS destination is open with `style=listing`. That is the style that is used if you do not open any other destination or specify any other style. You can open other destinations, e.g. HTML (by default `style=default`), printer (by default `style=printer`), RTF (by default `style=rtf`), and so on by specifying them in an ODS destination statement. Other styles of particular interest include `statistical`, `analysis`, and `journal`. There are many other styles as well. Each destination has a default style, and each permits a style specification. The following statement enables the HTML destination with the `statistical` style:

```
ods html body='b.html' style=statistical;
```

If listing (with `style=listing`) and one other destination are open, the macro will run using the style from the other destination. If other destinations are open as well, the macro will only run if there is one style other than `listing`. Hence, submitting the following statements before running the macro will cause the macro to issue an error and quit:

```
ods html;
ods rtf;
ods printer;
```

The `default`, `rtf` and `printer` styles are in effect, and the macro has no basis to choose between them. However, the macro will run fine with the following destinations since there is only one style other than the `listing` style in effect, and that is `style=default`:

```
ods html body='b.html';
ods rtf style=default;
ods printer style=default;
```

When you specify a destination other than the listing destination, you can usually make everything run faster by closing the listing destination, if you are not interested in results from that destination, for example, as follows:

```
ods listing close;
```

Note that a macro such as the `%PlotIt` macro only has limited and varying access to style information. For most styles, it gets all of the information it needs, however, for some styles, that information is just not there, so you might not always get all of the right colors such as background colors.

If you do not like the default background color, specify `gopplot=cback=`*some-color*, (substituting your favorite color for some-color). This controls the area outside the graph. The area inside the graph is controlled by the `cframe=` option. You can permanently change the default by modifying the macro. Similarly, you can change `color=` and `colors=` as you see fit. The following step creates a plot with a white background inside the axes and a light gray background around the axes:

```
%plotit(data=sashelp.class(drop=age sex), cframe=white, gopts=cback=ligr)
```

If you are specifying colors, you will probably want to use the `nogstyle` option so that the style does not override the colors that you specify.

Sample Usage

For many plots, you only need to specify the `data=` and `datatype=` options. The following steps perform a simple correspondence analysis, however, PROC CORRESP and ODS Graphics can automatically make the same plot:

```
*------Simple Correspondence Analysis------;
proc corresp all data=cars outc=coor;
   tables marital, origin;
   title 'Simple Correspondence Analysis';
   run;

%plotit(data=coor, datatype=corresp)
```

The following steps perform a multiple correspondence analysis, however, PROC CORRESP and ODS Graphics can automatically make the same plot:

```
*------Multiple Correspondence Analysis------;
proc corresp mca observed data=cars outc=coor;
   tables origin size type income home marital sex;
   title 'Multiple Correspondence Analysis';
   run;

%plotit(data=coor, datatype=mca)
```

The following steps perform a multidimensional preference analysis, however, PROC PRINQUAL and ODS Graphics can automatically make the same plot:

```
*------MDPREF------;
proc prinqual data=carpref out=results n=2 replace mdpref;
   id model mpg reliable ride;
   transform ide(judge1-judge25);
   title 'Multidimensional Preference (MDPREF) Analysis';
   run;

%plotit(data=results, datatype=mdpref 2.5)
```

The vector lengths are increased by a factor of 2.5 to make a better graphical display.

The following steps perform a preference mapping vector model, however, PROC TRANSREG and ODS Graphics can automatically make the same plot:

```
*------PREFMAP, Vector Model------;
proc transreg data=results(where=(_type_ = 'SCORE'));
   model ide(mpg reliable ride)=identity(prin1 prin2);
   output tstandard=center coefficients replace out=tresult1;
   id model;
   title 'Preference Mapping (PREFMAP) Analysis - Vector';
   run;

%plotit(data=tresult1, datatype=vector 2.5)
```

Again, the vector lengths are increased by a factor of 2.5 to make a better graphical display.

The following steps perform a preference mapping ideal point model, however, PROC TRANSREG and ODS Graphics can automatically make the same plot:

```
*------PREFMAP, Ideal Point------;
proc transreg data=results(where=(_type_ = 'SCORE'));
   model identity(mpg reliable ride)=point(prin1 prin2);
   output tstandard=center coordinates replace out=tresult1;
   id model;
   title 'Preference Mapping (PREFMAP) Analysis - Ideal';
   run;

%plotit(data=tresult1, datatype=ideal, antiidea=1)
```

The `antiidea=1` option is specified to handle anti-ideal points when large data values are positive or ideal.

The following steps perform multidimensional preference analysis, however, PROC PRINQUAL and ODS Graphics can automatically make the same plot:

```
*------MDPREF, labeled vector end points------;
proc prinqual data=recreate out=rec mdpref rep;
   transform identity(sub:);
   id activity active relaxing spectato;
   title 'MDPREF of Recreational Activities';
   run;

%plotit(data=rec, datatype=mdpref2 3,
        symlen=2, vechead=, adjust1=%str(
        if _type_ = 'CORR' then do;
           __symbol = substr(activity,4);
           __ssize  = 0.7;
           activity = ' ';
           end;))
```

The `mdpref2` specification means MDPREF and label the vectors *too*. The vector lengths are increased by a factor of 3 to make a better graphical display. The `symlen=2` option specifies two-character symbols. The specification `vechead=`, (a null value) means no vector heads since there are labels. The `adjust1=` option is used to add full SAS DATA step statements to the preprocessed data set. This example processes `_type_ = 'CORR'` observations (those that contain vector the coordinates) the original variable names (`sub1`, `sub2`, `sub3`, ..., from the `activity` variable) and creates symbol values (1, 2, 3, ...) of size 0.7. The result is a plot with each vector labeled with a subject number.

The following steps create a contour plot, displaying density with color:

```
*------Bivariate Normal Density Function------;
proc iml;
   title 'Bivariate Normal Density Function';
   s = inv({1 0.25 , 0.25 1});
   m = -2.5; n = 2.5; inc = 0.05; k = 0;
   x = j((1 + (n - m) / inc) ** 2, 3, 0);
   c = sqrt(det(s)) / (2 * 3.1415);
   do i = m to n by inc;
      do j = m to n by inc;
         v = i || j; z = c * exp(-0.5 * v * s * v`);
         k = k + 1; x[k,] = v || z;
         end;
      end;
   create x from x[colname={'x' 'y' 'z'}]; append from x;
   quit;

%plotit(datatype=contour, data=x, extend=close,
         paint=z white blue magenta red)
```

The `paint=z white blue magenta red` option specifies that color interpolation is based on the variable `z`, going from white (zero density) through blue, magenta, and to red (maximum density). This color list is designed for a background of white. The option `extend=close` is used with contour plots so that the plot boundaries appear exactly at the edge of the contour data. By default, `%PlotIt` usually adds a bit of extra white space between the data and the plot boundaries which provides extra room for labels, which are not used in this example. Similar plots can be made with ODS Graphics.

The goal of this next example is to create a plot of the Fisher iris data set with each observation identified by its species. Similar and nicer plots can be made with ODS Graphics. Species name is centered at each point's location, and each species name is plotted in a different color. This scatter plot is overlaid on the densities used by PROC DISCRIM to classify the observations. There are three densities, one for each species. Density is portrayed by a color contour plot with white (the assumed background color) indicating a density of essentially zero. Yellow, orange, and red indicate successively increasing density.

The `data=` option names the input SAS data set. The `plotvars=` option names the y-axis and x-axis variables. The `labelvar=_blank_` option specifies that all labels are blank. This example does not use any of PROC PLOT's label collision avoidance code. It simply uses PROC PLOT to figure out how big to make the plot, and then the macro puts everything inside the plot independently of PROC PLOT, so the printer plot is blank. The `symlen=4` option specifies that the maximum length of a symbol value is 4 characters. This is because we want the first four characters of the species names as symbols. The `exttypes=symbol contour` option explicitly specifies that PROC PLOT will know nothing about the symbols or the contours. They are external types that are added to the graphical plot by the macro after PROC PLOT has finished. The `ls=100` option specifies a constant line size. Since no label avoidance is done, there can be no collisions, and the macro will not iteratively determine the plot size. The default line size of 65 is too small for this example, whereas `ls=100` makes a better display. The `paint=` option specifies that based on values the variable `density`, colors should be interpolated ranging from white (minimum `density`) to yellow to orange to red (maximum `density`). The `rgbtypes=contour` option specifies that the `paint=` option should apply to contour type observations.

The grid (created with the loops: `do sepallen = 30 to 90 by 0.6;` and `do petallen = 0 to 80 by 0.6;`) is not square, so for optimal results the macro must be told the number of horizontal and vertical positions. The PLOTDATA DATA step creates these values and stores them in macro variables `&hnobs` and `&vnobs`, so the specification `hnobs=&hnobs, vnobs=&vnobs`, specifies the grid size. Of course these values could have been specified directly instead of through symbolic variables. The `excolors=CXFFFFFF` option is included for efficiency. The input data consist of a large grid for the contour plot. Most of the densities are essentially zero, so many of the colors are `CXFFFFFF`, which is white, computed by `paint=`, which is the same color as the background. (See the `paint=` option, page 1204 for information about CX*rrggbb* color specifications.) Excluding them from processing makes the macro run faster and creates smaller datasets.

This example shows how to manually do the kinds of things that the `datatype=` option does for you with standard types of data sets. The macro expects the data set to contain observations of one or more types. Each type is designated by a different value in a variable, usually named `_type_`. In this example, there are four types of observations, designated by the `_type_` variable's four values, 1, 2, 3, 4, which are specified in the `types=` option. The `symtype=` option specifies the symbol types for these four observation types. The first three types of observations are `symbol` and the last type, `_type_` = 4, designates the contour observations. The first three symbols are the species names (`symbols=` values) displayed in `symfont=none` (hardware font). The last symbol is null because contours do not use symbols. The first three symbols, since they are words as opposed to a single character, are given a small size (`symsize=0.6`). A value of 1 is specified for the symbol size for contour type observations. The macro determines the optimal size for each color rectangle of the contour plot. The following steps process the data, perform the analysis, and create the plot:

```
*------Discriminant Analysis------;
data plotdata;  * Create a grid over which DISCRIM outputs densities.;
   do SepalLength = 30 to 90 by 0.6;
      h + 1; * Number of horizontal cells;
      do PetalLength = 0 to 80 by 0.6;
         n + 1; * Total number of cells;
         output;
         end;
      end;
   call symput('hnobs', compress(put(h    , best12.))); * H grid size;
   call symput('vnobs', compress(put(n / h, best12.))); * V grid size;
   drop n h;
   run;

proc discrim data=iris testdata=plotdata testoutd=plotd
             method=normal pool=no short noclassify;
   class species;
   var PetalLength SepalLength;
   title 'Discriminant Analysis of Fisher (1936) Iris Data';
   title2 'Using Normal Density Estimates with POOL=NO';
   run;

data all;
   * Set the density observations first so the scatter plot points
     are on top of the contour plot.  Otherwise the contour plot
     points will hide the scatter plot points.;
   set plotd iris(in=iris);
   if iris then do;
      _type_ = species; * unformatted species number 1, 2, 3;
      output;
      end;
   else do;
      _type_ = 4; * density observations;
      density = max(setosa,versicolor,virginica);
      output;
      end;
   run;

%plotit(data=all, plotvars=PetalLength SepalLength, labelvar=_blank_,
        symlen=4, exttypes=symbol contour, ls=100, gopplot=cback=white,
        paint=density white yellow orange red, rgbtypes=contour,
        hnobs=&hnobs, vnobs=&vnobs, excolors=CXFFFFFF,
        types  =1      2          3          4,
        symtype=symbol symbol     symbol     contour,
        symbols=Set    Vers       Virg       '',
        symsize=0.6    0.6        0.6        1,
        symfont=none   none       none       solid
        )

title;
```

How %PlotIt Works

You create a data set either with a DATA step or with a procedure. Then you run the macro to create a graphical scatter plot. This macro is not a SAS/GRAPH procedure (although it uses PROC GANNO), and in many ways, it does not behave like a typical SAS/GRAPH procedure. The `%PlotIt` macro performs the following steps.

1. The `%PlotIt` macro reads an input data set and preprocesses it. The preprocessed data set contains information such as the axis variables, the point-symbol and point-label variables, and symbol and label types, sizes, fonts, and colors. The nature of the preprocessing depends on the type of data analysis that generated the input data set. For example, if the option `datatype=mdpref` had been specified with an input data set created by PROC PRINQUAL for a multidimensional preference analysis, then the `%PlotIt` macro creates blue points for `_type_ = 'SCORE'` observations and red vectors for `_type_ = 'CORR'` observations.
2. A DATA step, using the DATA Step Graphics Interface, determines how big to make the graphical plot.
3. PROC PLOT determines where to position the point labels. By default, if some of the point label characters are hidden, the `%PlotIt` macro recreates the printer plot with a larger line and page size, and hence creates more cells and more room for the labels. Note that when there are no point labels, the printer plot might be empty. All of the information that is in the graphical scatter plot can be stored in the `extraobs=` data set. All results from PROC PLOT are written to data sets with ODS. The macro will clear existing `ods select` and `ods exclude` statements.
4. The printer plot is read and information from it, the preprocessed data set, and the extra observations data set are combined to create an Annotate data set. The label position information is read from the PROC PLOT output, and all of the symbol, size, font, and color information is extracted from the preprocessed (or extra observations) data set. The Annotate data set contains all of the instructions for drawing the axes, ticks, tick marks, titles, point symbols, point labels, axis labels, and so on. Circles can be drawn around certain locations, and vectors can be drawn from the origin to other locations.
5. The Annotate data set is displayed with the GANNO procedure. The `%PlotIt` macro does not use PROC GPLOT.

Debugging

When you have problems, try `debug=vars` to see what the macro thinks you specified. It is also helpful to specify: `debug=mprint notes`. You can also display the final Annotate data set and the preprocessing data set, for example, as follows:

```
options ls=180;
proc print data=anno uniform;
   format text $20. comment $40.;
   run;

proc print data=preproc uniform;
   run;
```

Advanced Processing

You can post-process the Annotate DATA step to change colors, fonts, undesirable placements, and so on. Sometimes, this can be done with the `adjust4=` option. Alternatively, when you specify `method=none`, you create an Annotate data set without displaying it. The data set name is by default WORK.ANNO. You can then manipulate it further with a DATA step or PROC FSEDIT to change colors, fonts, or sizes for some labels; move some labels; and so on. If the final result is a new data set called ANNO2, you can display it by running the following:

```
proc ganno annotate=anno2;
   run;
```

Notes

With `method=print`, the macro creates a file. See the `filepref=` and `post=` options and make sure that the file name does not conflict with existing names.

This macro creates variable names that begin with two underscores and assumes that these names will not conflict with any input data set variable names.

It is not feasible with a macro to provide the full range of error checking that is provided with a procedure. Extensive error checking is provided, but not all errors are diagnosed.

Not all options will work with all other options. Some combinations of options might produce macro errors or Annotate errors.

This macro might not be fully portable. When you switch operating systems or graphics devices, some changes might be necessary to get the macro to run successfully again.

Graphics device differences might also be a problem. We do not know of any portability problems, but the macro has not been tested on all supported devices.

Some styles are structured in such a way that the `%PlotIt` macro cannot extract information from them. For those styles, default colors are used instead.

This macro tries to create a plot with equated axes, where a centimeter on one axis represents the same data range as a centimeter on the other axis. The only way to accomplish this is by explicitly and jointly controlling the `hsize=`, `vsize=`, `hpos=`, and `vpos=` `goptions`. By default, the macro tries to ensure that all of the values work for the specific device. See `makefit=`, `xmax=`, and `ymax=`. By default the macro uses GASK to determine `xmax` and `ymax`. If you change any of these options, your axes might not be equated. Axes are equated when `vsize` $\times$ `hpos` / `hsize` $\times$ `hpos` = `vtoh`.

When you are plotting variables that have very different scales, you might need to specify appropriate tick increments for both axes to get a reasonable plot, like this for example: `plotopts=haxis=by 20 vaxis=by 5000`. Alternatively, just specifying the smaller increment is often sufficient: `plotopts= haxis= by 20`. Alternatively, specify `vtoh=`, (null value) to get a plot like PROC GPLOT's, with the window filled.

By default, the macro iteratively creates and recreates the plot, increasing the line size and the flexibility in the `placement=` list until there are no penalties.

The SAS system option `ovp` (overprint) is not supported by this macro.

%PlotIt Macro Options

The following options can be used with the `%PlotIt` macro:

Option	Description
`help`	(positional) "help" or "?" displays syntax summary
`adjust1=`*SAS-statements*	adjust the preprocessing data set
`adjust2=`*SAS-statements*	includes statements with PROC PLOT
`adjust3=`*SAS-statements*	extra statements for the final DATA step
`adjust4=`*SAS-statements*	extra statements for the final DATA step
`adjust5=`*SAS-statements*	extra statements for the final DATA step
`antiidea=`*n*	eliminates PREFMAP anti-ideal points
`blue=`*expression*	blue part of RGB colors
`bright=`*n*	generates random label colors
`britypes=`*type*	types to which `bright=` applies
`cframe=`*color*	color of background within the frame
`cirsegs=`*n*	circle smoothness parameter
`color=`*color*	default color
`colors=`*colors-list*	default label and symbol color list
`cursegs=`*n*	number of segments in a curve
`curvecol=`*color*	color of curve
`data=`*SAS-data-set*	input data set
`datatype=`*data-type*	data analysis that generated data set
`debug=`*values*	debugging output
`excolors=`*color-list*	excludes from the Annotate data set
`extend=`*axis-extensions*	extend the x and y axes
`extraobs=`*SAS-data-set*	extra observations data set
`exttypes=`*type*	types for `extraobs=` data set
`filepref=`*prefix*	file name prefix
`font=`*font*	default font
`framecol=`*color*	color of frame
`gdesc=`*description*	catalog description
`gname=`*name*	catalog entry
`gopplot=`*goptions*	`goptions` for plotting to screen
`gopprint=`*goptions*	`goptions` for printing
`gopts2=`*goptions*	`goptions` that are always used
`gopts=`*goptions*	additional `goptions`
`gout=`*catalog*	`proc ganno gout=` catalog
`green=`*expression*	green part of RGB colors
`hminor=`*n* | *do-list*	horizontal axis minor tick marks
`hnobs=`*n*	horizontal observations for contour plots
`hpos=`*n*	horizontal positions in graphics area
`href=`*do-list*	horizontal reference lines
`hsize=`*n*	horizontal graphics area size
`inc=`*n*	`haxis=by inc`, `vaxis=by inc`
`interpol=`*method*	axis interpolation method
`labcol=`*label-colors*	colors for the point labels
`label=`*label-statement*	`label` statement
`labelcol=`*color*	color of variable labels

Option	Description
`labelvar=`*label-variable*	point label variable
`labfont=`*label-fonts*	fonts for the point labels
`labsize=`*label-sizes*	sizes for the point labels
`ls=`*n*	how line sizes are generated
`lsinc=`*n*	increment to line size
`lsizes=`*number-list*	line sizes (thicknesses)
`makefit=`*n*	proportion of graphics window to use
`maxiter=`*n*	maximum number of iterations
`maxokpen=`*n*	maximum acceptable penalty sum
`method=`*value*	where to send the plot
`monochro=`*color*	overrides all other colors
`nknots=`*n*	number of knots option
`offset=`*n*	move symbols for coincident points
`options=border`	draws a border box
`options=close`	close up the border box and the axes
`options=diag`	draws a diagonal reference line
`options=expand`	expands the plot to fill the window
`options=noback`	do not set the frame color
`options=nocenter`	do not center
`options=noclip`	do not clip
`options=nocode`	suppress the PROC PLOT and `goptions` statements
`options=nodelete`	do not delete intermediate data sets
`options=nohistory`	suppress the iteration history table
`options=nolegend`	suppress the display of the legends
`options=noprint`	`nolegend`, `nocode`, and `nohistory`
`options=square`	the same ticks for both axes and a square plot
`options=textline`	lines overwrite text
`out=`*SAS-data-set*	output Annotate data set
`outward=none` | *char*	PLOT statement `outward=`
`paint=`*interpolation*	color interpolation
`place=`*placement*	generates a `placement=` option
`plotopts=`*options*	PLOT statement options
`plotvars=`*variable-list*	y-axis and x-axis variables
`post=`*filename*	graphics stream file name
`preproc=`*SAS-data-set*	preprocessed `data=` data set
`procopts=`*options*	PROC PLOT statement options
`ps=`*n*	page size
`radii=`*do-list*	radii of circles
`red=`*expression*	red part of RGB colors
`regdat=`*SAS-data-set*	intermediate regression results data set
`regopts=`*options*	regression curve fitting options
`regprint=`*regression-option*	regression options
`rgbround=`*RGB-rounding*	`paint=` rounding factors
`rgbtypes=`*type*	types to which `paint=` and RGB options apply
`style=`A | B	obsolete option not supported
`symbols=`*symbol-list*	plotting symbols
`symcol=`*symbol-colors*	colors of the symbols
`symfont=`*symbol fonts*	symbol fonts

Option	Description
`symlen=`*n*	length of the symbols
`symsize=`*symbol-sizes*	sizes of symbols
`symtype=`*symbol-types*	types of symbols
`symvar=`*symbol-variable*	plotting symbol variable
`tempdat1=`*SAS-data-set*	intermediate results data set
`tempdat2=`*SAS-data-set*	intermediate results data set
`tempdat3=`*SAS-data-set*	intermediate results data set
`tempdat4=`*SAS-data-set*	intermediate results data set
`tempdat5=`*SAS-data-set*	intermediate results data set
`tempdat6=`*SAS-data-set*	intermediate results data set
`tickaxes=`*axis-string*	axes to draw tick marks
`tickcol=`*color*	color of ticks
`tickfor=`*format*	tick format used by `interpol=tick`
`ticklen=`*n*	length of tick mark in horizontal cells
`titlecol=`*color*	color of title
`tsize=`*n*	default text size
`types=`*observation-types*	observations types
`typevar=`*variable*	observation types variable
`unit=in` | `cm`	`hsize=` and `vsize=` unit
`vechead=`*vector-head-size*	how to draw vector heads
`vminor=`*n* | *do-list*	vertical axis minor tick marks
`vnobs=`*n*	vertical observations for contour plots
`vpos=`*n*	vertical positions in graphics area
`vref=`*do-list*	vertical reference lines
`vsize=`*n*	vertical graphics area size
`vtoh=`*n*	PROC PLOT `vtoh=` option
`xmax=`*n*	maximum horizontal graphics area size
`ymax=`*n*	maximum vertical graphics area size

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%plotit(help)
%plotit(?)
```

Note that for many analyses, the only options you need to specify are `data=`, `datatype=`, and sometimes `method=`. To specify variables to plot, specify `plotvars=`, `labelvar=`, and `symvar=`.

Overriding Options

This macro looks for a special global macro variable named `plotitop`. If it exists, its values are used to override the macro options. If you have a series of calls to the plotting macro and say, for example, that you want to route each graph to a PDF file, you can specify the following statement once:

```
%let plotitop = gopts=dev=pdf, method=print;
```

Then you can run the macro repeatedly without change. The value of the `plotitop` macro variable must consist of a name, followed by an equal sign, followed by a value. Optionally, it can continue with a comma, followed by another `name=`*value*, and so on, just like the way options are specified with the macro. Option values must not contain commas.

Destination and GOPTIONS

The options in this section specify the plot destination and SAS `goptions`. Note that with the `%PlotIt` macro, you do not specify a `goptions` statement. If you do, it will be overridden. All `goptions` (except `device=`) are specified with macro options. If you would prefer to specify your own `goptions` statement and have the macro use it, just specify or change the default for these four options to null: `gopplot=`, `gopprint=`, `gopts2=`, `gopts=`. If you use a locally installed copy of the macro, you can modify the `gopprint=` and `gopplot=` options defaults to include the devices that you typically use. Otherwise, the macro checks the `goptions` to get a device.

gopplot= *goptions*
specifies the `goptions` for directly plotting on the screen. There are no default goptions for `gopplot=`.

gopprint= *goptions*
specifies the `goptions` for printing (creating a graphics stream file). The default is `gopprint=gsfmode=replace gaccess=gsasfile gsfname=gsasfile`.

The following options show how you might modify the defaults for `gopprint=` and `gopplot=` option defaults to set default devices:

```
gopprint=gsfmode=replace gaccess=gsasfile gsfname=gsasfile device=png,
gopplot=cback=black device=win,
```

gopts= *goptions*
provides a way to specify additional `goptions` that are always used. There are no default goptions for `gopts=`. For example, to rotate to a landscape orientation with a black background color, specify `gopts=rotate cback=black`.

gopts2= *goptions*
specifies the `goptions` that are always used, no matter which `method=` is specified. The default is `gopts2=reset=goptions erase`.

method= `gplot` | `plot` | `print` | `none`
specifies where to send the plot. The default is `method=gplot`.

`gplot` – displays a graphical scatter plot on your screen using the `goptions` from `gopplot=`. The `gopplot=` option should contain the `goptions` that only apply to plots displayed on the screen.

`plot` – creates a printer plot only.

`print` – routes the plot to a graphics stream file, such as a postscript file, using the `goptions` from `gopprint=`. The `gopprint=` option should contain the `goptions` that only apply to hard-copy plots. Specify the file name with `post=`.

`none` – just creates the Annotate data set and sets up `goptions` using `gopplot=`.

Data Set and Catalog Options

These options specify the input SAS data set, output Annotate data set, and options for writing plots to files and catalogs.

data= *SAS-data-set*

specifies the input data set. The default input data set is the last data set created. You should always specify the `data=` option since the macro creates data sets that are not suitable for use as input.

filepref= *file-name-prefix*

specifies the file name prefix. The default is `filepref=sasplot`.

gdesc= *description*

specifies the name of a catalog description. This option can optionally be used with `proc ganno gout=` to provide the `description=`.

gname= *name*

specifies the name of a catalog entry. This option can optionally be used with `proc ganno gout=` to provide the `name=`.

gout= *catalog*

specifies the `proc ganno gout=` catalog. With `gout=gc.slides`, first specify: `libname gc '.'`; Then to replay, run: `proc greplay igout=gc.slides; run;` Note that replayed plots will not in general have the correct aspect ratio.

out= *SAS-data-set*

specifies the output Annotate data set. This data set contains all of the instructions for drawing the graph. The default is `out=anno`.

post= *filename*

specifies the graphics stream file name. The default name is constructed from the `filepref=` value and '`ps`' in a host-specific way.

Typical Options

These are some of the most frequently used options.

datatype= *data-type*
specifies the type of data analysis that generated the data set. This option is used to set defaults for other options and to do some preprocessing of the data.

When the data type is `corresp`, `mds`, `mca`, `row`, `column`, `mdpref`, `mdpref2`, `vector`, or `ideal`, the `label=typical` option is implied when `label=` is not otherwise specified. The default point label variable is the last character variable in the data set.

Some data types (`mdpref`, `vector`, `ideal`, `corresp`, `row`, `mca`, `column`, `mds`) expect certain observation types and set the `types=` list accordingly. For example, `mdpref` expects `_type_ = 'SCORE'` and `_type_ = 'CORR'` observations. The remaining data types do not expect any specific value of the `typevar=` variable. So if you do not get the right data types associated with the right observation types, specify `types=`, and specify the `types=` values in an order that corresponds to the order of the symbol types in the `Types Legend` table. Unlike `symtype=`, the order in which you specify `datatype=` values is irrelevant.

A null value (`datatype=`, the default) specifies no special processing, and the default plotting variables are the first two numeric variables in the data set. Specifying `corresp`, `mds`, `mca`, `row`, or `column` will set the default `plotvars` to `dim2` and `dim1`. Otherwise, when a nonnull value is specified, the default `plotvars` are `prin2` and `prin1`.

The various data types are as follows:

`datatype=column`
specifies a `proc corresp profile=column` analysis. Row points are plotted as vectors.

`datatype=contour`
draws solid color contour plots. When the number of row points is not the same as the number of column points in the grid, use `hnobs=` and `vnobs=` to specify the number of points. This method creates an `hnobs=` by `vnobs=` grid of colored rectangles. Each of the rectangles should touch all adjacent rectangles. This method works well with a regular grid of points. The `method=square` option is a good alternative when the data do not fall in a regular grid.

`datatype=corresp`
specifies an ordinary correspondence analysis.

`datatype=curve`
fits a curve through the scatter plot.

`datatype=curve2`
fits a curve through the scatter plot and tries to make the labels avoid overprinting the curve.

`datatype=function`
draws functions. Typically, no labels or symbols are drawn. This option has a similar effect to the PROC GPLOT `symbol` statement options `i=join v=none`.

`datatype=ideal`
specifies a PREFMAP ideal point model. See the `antiidea=`, `radii=`, and `cirsegs=` options.

`datatype=mca`
specifies a multiple correspondence analysis.

`datatype=mdpref`
specifies multidimensional preference analysis with vectors with blank labels. Note that `datatype=mdpref` can also be used for ordinary principal component analysis.

`datatype=mdpref2`
specifies MDPREF with vector labels (MDPREF and labels *too*).

`datatype=mds`
specifies multidimensional scaling.

`datatype=mds ideal`
specifies PREFMAP ideal point after the MDS.

`datatype=mds vector`
specifies PREFMAP after MDS.

`datatype=row`
specifies a `proc corresp profile=row` analysis. Column points are plotted as vectors.

`datatype=square`
plots each point as a solid square. The `datatype=square` option is useful as a form of contour plotting when the data do not form a regular grid. The `datatype=square` option, unlike `datatype=contour`, does not try to scale the size of the square so that each square will touch another square.

`datatype=symbol`
specifies an ordinary scatter plot.

`datatype=vector`
specifies a PREFMAP vector model.

`datatype=vector ideal`
specifies both PREFMAP vectors and ideal points.

For some `datatype=` values, a number can be specified after the name. This is primarily useful for biplot data sets produced by PROC PRINQUAL and PREFMAP data sets produced by PROC TRANSREG. This number specifies that the lengths of vectors should be changed by this amount. The number must be specified last. Examples: `datatype=mdpref 2`, `datatype=mds vector 1.5`.

The primary purpose of the `datatype=` option is to provide an easy mechanism for specifying defaults for the options in the next section (`typevar=` through `outward=`).

labelvar= *label-variable* | `_blank_`

specifies the variable that contains the point labels. The default is the last character variable in the data set. If `labelvar=_blank_` is specified, the macro will create a blank label variable.

options= *options-list*

specifies binary options. Specify zero, one, or more in any order. For example: `options=nocenter nolegend`.

`border`
`noborder`

draws a border box around the outside of the graphics area, like the border `goption`. The default depends on the style, and with typical styles, there is a border.

`close`
if a border is being drawn, perform the same adjustments on the border that are performed on the axes. This option is most useful with contour plots.

`diag`
draws a diagonal reference line.

`expand`
specifies Annotate data set post processing, typically for use with `extend=close` and contour plots. This option makes the plot bigger to fill up more of the window.

`noback`
specifies that `%PlotIt` should not set the frame color (the background color within the plot boundary).

`nocenter`
do not center. By default, when `nocenter` is not specified, `vsize=` and `hsize=` are set to their maximum values, and the `vpos=` and `hpos=` values are increased accordingly. The x and y coordinates are increased to positioning the plot in the center of the full graphics area.

`noclip`
do not clip. By default, when `noclip` is not specified, labels that extend past the edges of the plot are clipped. This option will not absolutely prevent labels from extending beyond the plot, particularly when sizes are greater than 1.

`nocode`
suppresses the display of the PROC PLOT and `goptions` statements.

`nodelete`
do not delete intermediate data sets.

`nohistory`
suppresses the display of the iteration history table.

`nolegend`
suppresses the display of the legends.

`noprint`
equivalent to `nolegend`, `nocode`, and `nohistory`.

`square`
uses the same ticks for both axes and tries to make the plot square by tinkering with the `extend=` option. Otherwise, ticks might be different.

`textline`
put text in the data set, followed by lines, so lines overwrite text. Otherwise text overwrites lines.

plotvars= *two-variable-names*

specifies the y-axis variable then the x-axis variable. To plot `dim2` and `dim3`, specify `plotvars=dim2 dim3`. The `datatype=` option controls the default variable list.

symlen= *n*

specifies the length of the symbols. By default, symbols are single characters, but the macro can center longer strings at the symbol location.

symvar= *symbol-variable* | `_symbol_`

specifies the variable that contains the plotting symbol for input to PROC PLOT. When `_symbol_` is specified, which is the default, the symbol variable is created, typically from the `symbols=` list, which might be constructed inside the macro. (Note that the variable `__symbol` is created to contain the symbol for the graphical scatter plot. The variables `_symbol_` and `__symbol` might or might not contain the same values.) Variables can be specified, and the first `symlen=` characters are used for the symbol. When a null value (`symvar=`) or a constant value is specified, the symbol from the printer plot is used (which is always length one, no matter what is specified for `symlen=`). To get PROC PLOT pointer symbols, specify `symvar='00'x`, (hex null constant). To center labels with no symbols, specify: `symbols='', place=0`.

Observation-Type List Options

Data sets for plotting can have different types of observations that are plotted differently. These options let you specify the types of observations, the variable that contains the observation types, and the different ways the different types should be plotted. For many types of analyses, these can all be handled easily with the `datatype=` option, which sets analysis-specific defaults for the list options. When you can, you should use `datatype=` instead of the list options. If you do use the list options, specify a variable, in `typevar=`, whose values distinguish the observation types. Specify the list of valid types in `types=`. Then specify fonts, sizes, and so on for the various observation types. Alternatively, you can use these options with `datatype=`. Specify lists for just those label or symbol characteristics you want to change, for example, colors, fonts or sizes.

The lists do not all have to have the same number of elements. The number of elements in `types=` determines how many of the other list elements are used. When an observation type does not match one of the `type=` values, the results are the same as if the first type is matched. If one of the other lists is shorter than the `types=` list, the shorter list is extended by adding copies of the last element to the end. Individual list values can be quoted, but quotes are not required. *Embedded blanks are not permitted. If you embed blanks, you will not get the right results.* Values of the `typevar=` variable are compressed before they are used, so for example, an `_type_` value of `'M COEFFI'` must be specified as `'MCOEFFI'`.

britypes= *type*

specifies the types to which `bright=` applies. The default is `britypes=symbol`.

colors= *colors-list*

specifies the default color list for the `symcol=` and `labcol=` options. The default depends on the style. With `gstyle`, this list is ignored and the list comes from the style. Otherwise, the default is `colors=blue red green cyan magenta orange gold lilac olive purple brown gray rose violet salmon yellow`. This is the old legacy default. Colors such as these from the default ODS style `colors=CX2A25D9 CXB2182B CX01665E CX9D3CDB CX543005 CX2597FA CX7F8E1F CXB26084 CXD17800 CX47A82A CXB38EF3 CXF9DA04` tend to look much better than the default colors. (See page 1204 for information about CX*rrggbb* color specifications.)

exttypes= *type*

specifies the types to always put in the `extraobs=` data set when they have blank labels. The default is `exttypes=vector`.

labcol= *label-colors*

specifies the colors for the point labels. The default either comes from the style or from `colors=` with `nogstyle`. This option is ignored with `gstyle`. Examples (none means hardware):

```
labcol='red'
labcol='red' 'white' 'blue'
```

labfont= *label-fonts*

specifies the fonts for the point labels. Examples (none means hardware):

```
labfont=none
labfont='swiss'
labfont='swiss' 'swissi'
```

labsize= *label-sizes*

specifies the sizes for the point labels. Examples:

```
labsize=1
labsize=1 1.5
labsize=1 0
```

rgbtypes= *type*

specifies the types to which `paint=`, `red=`, `green=`, and `blue=` apply. The default is `rgbtypes=symbol`.

symbols= *symbol-list*

specifies the plotting symbols. Symbols can be more than a single character. You must specify `symlen=n` for longer lengths. Blank symbols must be specified as `''` with no embedded blanks. Examples:

```
symbols='*'
symbols='**'
symbols='*' '+' '*' ''
symbols='NC' 'OH' 'NJ' 'NY'
```

symcol= *symbol-colors*

specifies the colors of the symbols. The default list is constructed from the `colors=` option. Examples:

```
symcol='red'
symcol='red' 'white' 'blue'
```

symtype= *symbol-types*

specifies the types of symbols. Valid types include: `symbol`, `vector`, `circle`, `contour`, and `square`. Examples:

```
symtype='symbol'
symtype='symbol' 'vector'
symtype='symbol' 'circle'
```

symfont= *symbol fonts*

specifies the symbol fonts. The font is ignored for vectors with no symbols. Examples (none means hardware):

```
symfont=none
symfont='swiss'
symfont='swiss' 'swissi'
```

symsize= *symbol-sizes*

specifies the sizes of symbols. Examples:

```
symsize=1
symsize=1 1.5
```

types= *observation-types*

specifies the observations types. Observation types are usually values of a variable like `_type_`. Embedded blanks are not permitted. Examples:

```
types='SCORE'
types='OBS' 'SUPOBS' 'VAR' 'SUPVAR'
types='SCORE'
types='SCORE' 'MCOEFFI'
```

The order in which values are specified for the other options depends on the order of the types. The default types for various `datatype=` values are given next:

```
corresp:  'VAR' 'OBS' 'SUPVAR' 'SUPOBS'
row:      'VAR' 'OBS' 'SUPVAR' 'SUPOBS'
mca:      'VAR' 'OBS' 'SUPVAR' 'SUPOBS'
column:   'VAR' 'OBS' 'SUPVAR' 'SUPOBS'
mdpref:   'SCORE' 'CORR'
vector:   'SCORE' 'MCOEFFI'
ideal:    'SCORE' 'MPOINT'
mds:      'SCORE' 'CONFIG'
```

For combinations of options, these lists are combined in order, but without repeating `'SCORE'`, for example, with `datatype=mdpref vector ideal`, the default `types=` list is: `'SCORE' 'CORR' 'MCOEFFI' 'MPOINT'`.

typevar= *variable*
specifies a variable that is looked at for the observation types. By default, this is `_type_` if it is in the input data set.

Internal Data Set Options

The macro creates one or more of these data sets internally to store intermediate results.

extraobs= *SAS-data-set*
specifies a data set used to contain the extra observations that do not go through PROC PLOT. The default is `extraobs=extraobs`.

preproc= *SAS-data-set*
specifies a data set used to contain the preprocessed `data=` data set. The default is `preproc=preproc`.

regdat= *SAS-data-set*
specifies a data set used to contain intermediate regression results for curve fitting. The default is `regdat=regdat`.

tempdat1= *SAS-data-set*
specifies a data set used to hold intermediate results. The default is `tempdat1=tempdat1`.

tempdat2= *SAS-data-set*
specifies a data set used to hold intermediate results. The default is `tempdat2=tempdat2`.

tempdat3= *SAS-data-set*
specifies a data set used to hold intermediate results. The default is `tempdat3=tempdat3`.

tempdat4= *SAS-data-set*
specifies a data set used to hold intermediate results. The default is `tempdat4=tempdat4`.

tempdat5= *SAS-data-set*
specifies a data set used to hold intermediate results. The default is `tempdat5=tempdat5`.

tempdat6= *SAS-data-set*
specifies a data set used to hold intermediate results. The default is `tempdat6=tempdat6`.

Miscellaneous Options

The following options are sometimes needed for certain situations to control the details of the plots:

antiidea= *n*

eliminates PREFMAP anti-ideal points. The TRANSREG ideal-point model assumes that small attribute ratings mean that the object is similar to the attribute and large ratings imply dissimilarity to the attribute. For example, if the objects are food and the attribute is "sweetness," then the analysis assumes that 1 means sweet and 9 is much less sweet. The resulting coordinates are usually ideal points, representing an ideal amount of the attribute, but sometimes they are anti-ideal points and need to be converted to ideal points. This option is used to specify the nature of the data (small ratings mean similar or dissimilar) and to request automatic conversion of anti-ideal points.

null value – (`antiidea=`, the default) – do nothing.

1 – reverses in observations whose `_TYPE_` contains 'POINT' when `_issq_` > 0. Specify `antiidea=1` with `datatype=ideal` for the unusual case when large data values are positive or ideal.

–1 – reverses in observations whose `_type_` contains 'POINT' when `_issq_` < 0. Specify `antiidea=-1` with `datatype=ideal` for the typical case when small data values are positive or ideal.

extend= *axis-extensions*

is used to extend the x and y axes beyond their default lengths. Specify four values, for the left, right, top, and bottom axes. If the word `close` is specified somewhere in the string, then macro moves the axes in close to the extreme data values, and the computed values are added to the specified values (if any). Sample specifications: `extend=2 2`, or `extend=3 3 -0.5 0.5`. Specifying a positive value n extends the axis n positions in the indicated direction. Specifying a negative value shrinks the axis. The defaults are in the range –2 to 2, and are chosen in an attempt to add a little extra horizontal space and make equal the extra space next to each of the four extreme ticks. When there is enough space, the horizontal axis is slightly extended by default to decrease the chance of a label slightly extending outside the plot. PROC PLOT usually puts one or two more lines on the top of the plot than in the bottom. The macro tries to eliminate this discrepancy. This option does not add any tick marks; it just extends or shrinks the ends of the axis lines. So typically, only small values should be specified. Be careful with this option and a positive `makefit=` value.

font= *font*

specifies the default font. With `nogstyle`, the default is `font=swiss`. Otherwise the font comes from the style. The annotate data set will often contain fonts of `none`, which means that a hardware font is used. This will usually look much better than a software font such as the Swiss fonts that were heavily used before the 9.2 release.

hminor= *n* | *do-list*

specifies the number of horizontal axis minor tick marks between major tick marks. A typical value is 9. The number cannot be specified when `haxis=` is specified with `plotopts=`. Alternatively, specify a DATA step `do` list. Note that with log scaling, specify $\log_{10}$'s of the data values. For example, specify `hminor=0.25 to 5 by 0.25`, with data ranging up to 10^5.

href= *do-list*
specifies horizontal-axis reference lines (which are drawn vertically). Specify a DATA step `do` list. By default, there are no reference lines.

inc= *n*
specifies `haxis=by inc` and `vaxis=by inc` values. The specified increments apply to both axes. To individually control the increments, you must specify the PLOT statement `haxis=` and `vaxis=` options on the `plotopts=` option. When you are plotting variables that have very different scales, you might need to independently specify appropriate tick increments for both axes to get a reasonable plot, like this for example: `plotopts=haxis=by 20 vaxis=by 5000`.

interpol= `ls` | `tick` | `no` | `hlog` | `vlog` | `yes`
specifies the axis interpolation method.

> `ls` – uses the least-squares method only. This method computes the mapping between data and positions using ordinary least-squares linear regression. Usually, you should not specify `interpol=ls` because slight inaccuracies can result, producing aesthetically unappealing plots.
>
> `hlog` – specifies that the x-axis is on a log scale.
>
> `no` – does not interpolate.
>
> `tick` – uses the tick mark method. This method computes the slope and intercept using tick marks and their values. Tick marks are read using the `tickfor=` format.
>
> `vlog` – specifies that the y-axis is on a log scale.
>
> `yes` – the default, interpolates symbol locations, starting with least squares but replacing them with tick-based estimates when they are available.

This option makes the symbols, vectors, and circles map to the location they would in a true graphical scatter plot, not the cell locations from PROC PLOT. This option has no effect on labels, the frame, reference lines, titles, or ticks. With `interpol=no`, plots tend to look nicer whereas `interpol=yes` plots are slightly more accurate. Note that the strategy used to interpolate can be defeated in certain cases. If the horizontal axis tick values are displayed vertically, specify `interpol=ls`. The `hlog` and `vlog` values are specified in addition to the method. For example, `interpol=yes vlog hlog`.

label= *label-statement*
specifies a `label` statement. Note that specifying the keyword `label` to begin the statement is optional. You can specify `label=typical` to request a label statement constructed with `'Dimension'` and the numeric suffix of the variable name, for example, `label dim1 = 'Dimension 1' dim2 = 'Dimension 2';` when `plotvars=dim2 dim1`. The `label=typical` option can only be used with variable names that consist of a prefix and a numeric suffix.

ls= *n* | *iterative-specification*
specifies how line sizes are generated. The default is `ls=compute search`. When the second word is `search`, the macro searches for an optimal line size. See the `place=` option for more information about searches. When the first word is `compute`, the line size is computed from the iteration number so that the line sizes are: 65 80 100 125 150 175 200. Otherwise the first word is the first linesize and with each iteration the linesize is incremented by the `lsinc=` amount. Example: `ls=65 search`.

lsinc= *n*

specifies the increment to line size in iterations when line size is not computed. The default is `lsinc=15`.

lsizes= *number-list*

specifies the line sizes (thicknesses) for frame, ticks, vectors, circles, curves, respectively. The default is `lsizes=1 1 1 1 1`.

maxiter= *n*

specifies the maximum number of iterations. The default is `maxiter=15`.

maxokpen= *n*

specifies the maximum acceptable penalty sum. The default is `maxokpen=0`. Penalties accrue when label characters collide, labels get moved too far from their symbols, or words get split.

offset= *n*

move symbols for coincident points `offset=` spaces up/down and left/right. This helps to better display coincident symbols. Specify a null value (`offset=,`) to turn off offsetting. The default is `offset=0.25`.

place= *placement-specification*

generates a `placement=` option for the plot request. The default is `place=2 search`. Specify a non-negative integer. Values greater than 13 are set to 13. As the value gets larger, the procedure is given more freedom to move the labels farther from the symbols. The generated placement list is displayed in the log. You can still specify `placement=` directly in the `plotopts=` option. This option just gives you a shorthand notation. The following list shows the correspondence between these two options:

```
place=0 --placement=((s=center))


place=1 --placement=((h=2 -2 : s=right left)
                     (v=1 * h=0 -1 to -2 by alt))


place=2 --placement=((h=2 -2 : s=right left)
                     (v=1 -1 * h=0 -1 to -5 by alt))


place=3 --placement=((h=2 -2 : s=right left)
                     (v=1 to 2 by alt * h=0 -1 to -10 by alt))


place=4 --placement=((h=2 -2 : s=right left)
                     (v=1 to 2 by alt * h=0 -1 to -10 by alt)
                     (s=center right left * v=0 1 to 2 by alt *
                      h=0 -1 to -6 by alt * l= 1 to 2))
```

and so on.

The `place=` option, along with the `ls=` option can be used to search for an optimal placement list and an optimal line size. By default, the macro will create and recreate the plot until it avoids all collisions.

The search is turned off when a `placement=` option is detected in the plot request or plot options.

If search is not specified with `place=` or `ls=`, the specified value is fixed. If search is specified with the other option, only that option's value is incremented in the search.

plotopts= *options*
specifies PLOT statement options. The `box` option is specified, even if you do not specify it. Reference lines should not be specified using the PROC PLOT `href=` and `vref=` options. Instead, they should be specified directly using the `href=` and `vref=` macro options. By default, no PLOT statement options are specified except `box`.

procopts= *options*
specifies PROC PLOT statement options. The default is `procopts=nolegend`.

tickaxes= *axis-string*
specifies the axes to draw tick marks. The default with `nogstyle` is `tickaxes=LRTBFlb`, and with a style, the default is `tickaxes=LBF`, which provides a cleaner look than the old default, more like ODS Graphics. The specification, `tickaxes=LRTBFlb`, means major ticks on left (L), right (R), top (T), and bottom (B), and the full frame (F) is to be drawn, and potentially minor tick marks on the left (l) and bottom (b). Minor ticks on the right (r) and top (t) can also be requested. To just have major tick marks on the left and bottom axes, and no full frame, specify `tickaxes=LB`. Order and spacing do not matter. `hminor=` and `vminor=` must also be specified to get minor ticks.

tickfor= *format*
specifies the tick format used by `interpol=tick`. You should change this if the tick values in the PROC PLOT output cannot be read with the default `tickfor=32.` format. For example, specify `tickfor=date7.` with dates.

ticklen= *n*
specifies the length of tick marks in horizontal cells. A negative value can be specified to indicate that only half ticks should be used, which means the ticks run to but not across the axes. The default is `ticklen=1.5`.

tsize= *n*
specifies the default text size. The default is `tsize=1`.

vminor= *n* | *do-list*
specifies the number of vertical axis minor tick marks between major tick marks. A typical value is 9. The number cannot be specified when `vaxis=` is specified with `plotopts=`. Alternatively, specify a DATA step `do` list. Note that with log scaling, specify $\log_{10}$'s of the data values. For example, specify `vminor=0.25 to 5 by 0.25`, with data ranging up to 10^5.

vref= *do-list*

specifies vertical reference lines (which are drawn horizontally). Specify a DATA step `do` list. By default, there are no reference lines.

Color Options

With `gstyle`, the symbol and point label colors are set by the style. Otherwise, the symbol and point label colors are set by the `labcol=` and `symcol=` options. The other color options are as follows:

bright= *n*

generates random label colors for `britypes=` values. In congested plots, it might be easier to see which labels and symbols go together if each label/symbol pair has a different random color. Colors are computed so that the mean RGB (red, green, blue) components equal the specified `bright=` value. The valid range is $5 \leq bright \leq 250$. 128 is a good value. Small values will produce essentially black labels and large values will produce essentially white labels, and so should be avoided. The default is a null value, `bright=`, and there are no random label colors. If you get a color table full error message, you need to specify larger values for the `rgbround=` option.

cframe= *color*

specifies the color of the background within the frame. This is analogous to the `cframe=` SAS/GRAPH option. With `gstyle`, the style color is used. This option is ignored with `gstyle`.

color= *color*

specifies the default color that is used when no other color is set. The default color is black. With `gstyle`, this option will typically have little or no effect.

curvecol= *color*

specifies the color of curves in a regression plot. The default either comes from the style or from `color=` with `nogstyle`. This option is ignored with `gstyle`.

excolors= *color-list*

excludes observations from the Annotate data set with colors in this list. For example, with a white background, to exclude all observations that have a color set to white as well as those with a computed white color, for example, from `bright=` or `paint=`, specify `excolors=white CXFFFFFF`. This is done for efficiency, to make the Annotate data set smaller. (See the `paint=` option, page 1204 for information about CX*rrggbb* color specifications.)

framecol= *color*

specifies the color of the frame. The default either comes from the style or from `color=` with `nogstyle`. This option is ignored with `gstyle`.

labelcol= *color*

specifies the color of the variable labels. The default either comes from the style or from `color=` with `nogstyle`. This option is ignored with `gstyle`.

monochro= *color*
overrides all other specified colors. By default when `monochro=` is null, this option has no effect. Typical usage: `monochro=black`. Typically, you would rarely if ever use this option, and you would only use it with `nogstyle`.

style= this option is obsolete and is no longer supported. Use OPTIONS `gstyle/nogstyle` and a `style=` specification in your ODS destination to control the style. The `nogstyle` option will make the macro run the way it did in older releases. The `gstyle` option will make the appearance sensitive to the ODS style. The new results are usually superior. However, if you want explicit color control, you will typically need to specify `nogstyle`.

tickcol= *color*
specifies the color of tick labels. The default either comes from the style or from `color=` with `nogstyle`. This option is ignored with `gstyle`.

titlecol= *color*
specifies the color of the title. The default either comes from the style or from `color=` with `nogstyle`. This option is ignored with `gstyle`.

Color Interpolation and Painting

These next options are used to create label and symbol colors using some function of the input data set variables. For example, you can plot the first two principal components on the x and y axes and show the third principal component in the same plot by using it to control the label colors. The `paint=` option gives you a simple and fairly general way to interpolate colors. The `red=`, `green=`, and `blue=` options are used together for many other types of interpolations, but these options are much harder to use. These options apply to `rgbtypes=` observations. If `red=`, `green=`, and `blue=` are not flexible enough, for example, if you need full statements, specify `red=128` (so later code will know you are computing colors) then insert the full statements you need to generate the colors using `adjust1=`.

paint= *color-interpolation-specification*
is used to interpolate between colors based on the values of a variable. The simplest specification is `paint=variable`. More generally, specify the following option:

```
paint=variable optional-color-list optional-data-value-list
```

The following color names are recognized: red, green, blue, yellow, magenta, cyan, black, white, orange, brown, gray, olive, pink, purple, violet. For other colors, specify the RGB color name. The `paint=` option applies to the observation types mentioned in the `rgbtypes=` option. Valid types include: `symbol`, `vector`, `circle`, `contour`, and `square`.

Colors can be represented as `CX`*rrggbb* where *rr* is the red value, *gg* is the green, and *bb* is blue, all three specified in hex. The base ten numbers 0 to 255 map to 00 to FF in hex. For example, white is `CXFFFFFF` (all colors at their maximum), black is `CX000000` (all colors at their minimum), red is `CXFF0000` (maximum red, minimum green and blue), blue is `CX0000FF` (maximum blue, minimum red and green), and magenta is `CXFF00FF` (maximum red and blue, minimum green). When a variable named `z` is specified with no other arguments, the default is `paint=z blue magenta red`. The option `paint=z red green 1 10` interpolates between red and green, based on the values of the variable `z`, where values of 1 or less map to red, values of 10 or more map to green, and values in between map to colors in between. The specification `paint=z red yellow green 1 5 10`, interpolates between red at `z=1`, yellow at `Z=5`, and green at `Z=10`. If the data value list is omitted, it is computed from the data.

red= *expression*
green= *expression*
blue= *expression*
specify for arithmetic expressions that produce integers in the range 0 to 255. Colors are created as follows:

```
__color = 'CX' ||
          put(%if &red   ne %then round(&red,  __roured); %else 128; ,hex2.) ||
          put(%if &green ne %then round(&green,__rougre); %else 128; ,hex2.) ||
          put(%if &blue  ne %then round(&blue, __roublu); %else 128; ,hex2.);}
```

The `__rou` variables are extracted from the second through fourth values of the `rgbround=` option. Example: `red = min(100 + (z - 10) * 3, 255)`, `blue=50`, `green=50`. Then all labels are various shades of red, depending on the value of `z`. Be aware that light colors (small red-green-blue values) do not show up well on white backgrounds and dark colors do not show up well on dark backgrounds. Typically, you will not want to use the full range of possible red-green-blue values. Computed values greater than 255 are set to 255.

rgbround= *RGB-rounding-specification*
specifies rounding factors used for the `paint=` variable and RGB values. The default is `rgbround=-240 1 1 1`. The first value is used to round the `paint=` variable. Specify a positive value to have the variable rounded to multiples of that value. Specify a negative value $-n$ to have a maximum of n colors. For the other three values, specify positive values. The last three are rounding factors used to round the values for the red, green, and blue component of the color (see `red=`). If more than 256 colors are generated, you will get the error that a color was not added because the color table is full. By default, when a value is missing, there is no rounding. Rounding the `paint=` variable is useful with contour plots.

Contour Options

Use these options with contour plots. For example, the grid for a contour plot might be generated as follows:

```
do x = -4 to 4 by 0.1;
   do y = -2 to 2 by 0.1;
      ... statements ...
      end;
   end;
```

Horizontally, there are $1+(4--4)/0.1 = 81$ horizontal points and $1+(2--2)/0.1 = 41$ vertical points, so you would specify `hnobs=81`, `vnobs=41`. By default, the square root of the number of contour type observations is used for both `hnobs=` and `vnobs=` (which assumes a square grid).

hnobs= *n*
specifies the number of horizontal observations in the grid for contour plots.

vnobs= *n*
specifies the number of vertical observations in the grid for contour plots.

Advanced Plot Control Options

You can use the these next options to add full SAS DATA step statements to strategic places in the macro, such as the PROC PLOT step, the end of the preprocessing, and last full DATA steps. These options do minor adjustments before the final plot is produced. These options allow very powerful customization of your results to an extent not typically found in procedures. However, they might require a fair amount of work and some trial and error to understand and get right.

adjust1= *SAS-statements*
specifies preprocessing SAS statements. The following variables are created in the preprocessing data set:

`__lsize` – label size
`__lfont` – label font
`__lcolor` – label color
`__ssize` – symbol size
`__sfont` – symbol font
`__scolor` – symbol color
`__stype` – symbol type
`__symbol` – symbol value
`__otype` – observation type

Use `adjust1=` to adjust these variables in the preprocessing data set. You must specify complete statements with semicolons, for example, as follows:

```
   adjust1=%str(__lsize = 1.2; __lcolor = green;)

   adjust1=%str(if z > 20 then do;
   __scolor = 'green'; __lcolor = 'green'; end;)
```

adjust2= *SAS-statements*
specifies statements with PROC PLOT such as `format` statements. Just specify the full statement.

adjust3= *SAS-statements*
adjust4= *SAS-statements*
specify options to adjust the final Annotate data set. For example, in Swiss fonts, asterisks are not vertically centered when displayed, so `adjust3=` converts to use the SYMBOL function, so by default, `adjust3=%str(if text = '*' and function = 'LABEL' then do; style = ' '; text = 'star'; function = 'SYMBOL'; end;)`. The default for `adjust4=` is null, so you can use it to add new statements. If you add new variables to the data set, you must also include a `keep` statement.

The following illustrates using `adjust4=` to vertically display the y-axis label, like it would in PROC PLOT:

```
adjust4=%str(if angle = 90 then do; angle = 270; rotate = 90; keep rotate; end;)
```

The following option changes the size of title lines:

```
adjust4=%str(if index(comment, 'title') then size = 2;)
```

adjust5= *SAS-statements*

adds extra statements to the final DATA step that is used only for `datatype=function`. For example, to periodically mark the function with pluses, specify the following:

```
adjust5=%str(if mod(_n_,30) = 0 then do;
                size=0.25; function = 'LABEL'; text = '+'; output; end;)
```

Other Options

The remaining options for the `%PlotIt` macro are as follows:

cirsegs= *n*

specifies a circle smoothness parameter used in determining the number of line segments in each circle. Smaller values create smoother circles. The `cirsegs=` value is approximately related to the length of the line segments that compose the circle. The default is `cirsegs=.1`.

cursegs= *n*

specifies the number of segments in a regression function curve. The default is `cursegs=200`.

debug= vars | dprint | notes | time | mprint

specifies values that control debugging output.

`dprint` – print intermediate data sets.

`mprint` – run with `options mprint`.

`notes` – do not specify `options nonotes` during most of the macro.

`time` – displays total macro run time, ignored with `options nostimer;`

`vars` – displays macro options and macro variables for debugging.

You should provide a list of names for more than one type of debugging. Example: `debug=vars dprint notes time mprint`. The default is `debug=time`.

hpos= *n*

specifies the number of horizontal positions in the graphics area.

hsize= *n*

specifies the horizontal graphics area size in `unit=` units. The default is the maximum size for the device. By default, when `options=nocenter` is not specified, `hsize=` affects the size of the plot but not the `hsize= goption`. When `options=nocenter` is specified, `hsize=` affects both the plot size and the `hsize= goption`. If you specify just the `hsize=` but not `vsize=`, the vertical size is scaled accordingly.

makefit= *n*

specifies the proportion of the graphics window to use. When the `makefit=` value is negative, the absolute value is used, and the final value might be changed if the macro thinks that part of the plot might extend over the edge. When a positive value is specified, it will not be changed by the macro. When nonnull, the macro uses GASK to determine the minimum and maximum graphics window sizes and makes sure the plot can fit in them. The macro uses `gopprint=` or `gopplot=` to determine the device. The default is `makefit=-0.95`.

nknots= *n*

specifies the PROC TRANSREG number of knots option for regression functions.

outward= none | *char*

specifies a quoted string for the PLOT statement `outward=` option. Normally, this option's value is constructed from the symbol that holds the place for vectors. Specify `outward=none` if you want to not have `outward=` specified for vectors. The `outward=` option is used to greatly increase the likelihood that labels from vectors are displayed outward—away from the origin.

ps= *n*

specifies the page size.

radii= *do-list*

specifies the radii of circles (in a DATA step do list). The unit corresponds to the horizontal axis variable. The `radii=` option can also specify a variable in the input data set when radii vary for each point. By default, no circles are drawn.

regopts= *options*

specifies the PROC TRANSREG options for curve fitting. Example: `regopts=nknots=10 evenly`.

regfun= *regression-function*

specifies the function for curve fitting. Possible values include:

`linear` – line

`spline` – nonlinear spline function, perhaps with knots

`mspline` – nonlinear but monotone spline function, perhaps with knots

`monotone` – nonlinear, monotone step function

See PROC TRANSREG documentation for more information

regprint= *regression-options*
specifies the PROC TRANSREG PROC statement options, typically display options such as:

`noprint` – no regression printed output

`short` – suppress iteration histories

`ss2` – regression results

To see the regression table, specify: `regprint=ss2 short`. The default is `regprint=noprint`.

unit= `in` | `cm`
specifies the `hsize=` and `vsize=` unit in inches or centimeters (in or cm). The default is `unit=in`.

vechead= *vector-head-size* specifies how to draw vector heads. For example, the default specification `vechead=0.1 0.025`, specifies a head consisting of two hypotenuses from triangles with sides 0.1 units long along the vector and 0.025 units on the side perpendicular to the vector. This is smaller than the default in previous releases.

vpos= *n*
specifies the number of vertical positions in the graphics area.

vsize= *n*
specifies the vertical graphics area size in `unit=` units. The default is the maximum size for the device. By default when `options=nocenter` is not specified, `vsize=` affects the size of the plot but not the `vsize= goption`. When `options=nocenter` is specified, `vsize=` affects both the plot size and the `vsize= goption`. If you specify just the `vsize=` but not `hsize=`, the horizontal size is scaled accordingly.

vtoh= *n*
specifies the PROC PLOT `vtoh=` option. The `vtoh=` option specifies the ratio of the vertical height of a typical character to the horizontal width. The default is `vtoh=2`. Do not specify values much different than 2, especially by default when you are using proportional fonts. There is no one-to-one correspondence between characters and cells and character widths vary, but characters tend to be approximately twice as high as they are wide. When you specify `vtoh=` values larger than 2, near-by labels might overlap, even when they do not collide in the printer plot. The macro uses this option to equate the axes so that a centimeter on one axis represents the same data range as a centimeter on the other axis. A null value can be specified, `vtoh=`, when you want the macro to just fill the window, like a typical GPLOT.

Smaller values give you more lines and smaller labels. The specification `vtoh=1.75` is a good alternative to `vtoh=2` when you need more lines to avoid collisions. The specification `vtoh=1.75` means 7 columns for each 4 rows between ticks (7 / 4 = 1.75). The `vtoh=2` specification means the plot will have 8 columns for each 4 rows between ticks. Note that PROC PLOT sometimes takes this value as a hint, not as a rigid specification so the actual value might be slightly different, particularly when a value other than 2.0 is specified. This is generally not a problem; the macro adjusts accordingly.

xmax= *n*
specifies the maximum horizontal size of the graphics area.

ymax= *n*
specifies the maximum vertical size of the graphics area.

Macro Error Messages

Usually, if you make a mistake in specifying macro options, the macro will display an informative message and quit. These macros go to great lengths to check their input and issue informative error messages. However, *complete* error checking like we have with procedures is impossible in macros, and on rare occasions you might get a cascade of less than helpful error messages.[*] In that case, you will have to check the input and hunt for errors. One of the more common user errors is not providing a comma between options. Sometimes, for harder errors, specifying `options mprint;` will help you locate the problem. You might get a listing with a *lot* of code, almost all of which you can ignore. Search for the error and look at the code that comes before the error for ideas about what went wrong. Once you think you know which option is involved, be sure to also check the option before and after in your macro invocation, because that might be where the problem really is.

The `%PHChoice` macro uses PROC TEMPLATE and ODS to create customized output tables. Typically, the instructions for this customization, created by PROC TEMPLATE, are stored in a file under the `sasuser` directory with a host dependent name. On some hosts, this name is `templat.sas7bitm`. On other hosts, the name is some variation of the name `templat`. Sometimes this file can be corrupted. When this happens, the macro will not run correctly, and you will see error messages including errors about invalid pages. The solution is to find the corrupt file under `sasuser` and delete it (using your ordinary operating system file deletion method). After that, this macros should run fine again. If you have run any other PROC TEMPLATE customizations, you will need to rerun them after deleting the file. For more information, see “Template Store” or “Item Store” in the SAS ODS documentation.

Sometimes, when you run the `%MktEx` macro, you might see the following error:

```
ERROR: The MKTDES macro ended abnormally.
```

This is typically caused by one or more PROC FACTEX steps failing to find the requested design. When this happens, the macro recovers and continues searching. The macro does not always know in advance if PROC FACTEX will succeed. The only way for it to find out is by trying. The macro suppresses the PROC FACTEX error messages along with most other notes and warnings that would ordinarily come out. This error can be ignored.

Other times, when you run the `%MktEx` macro, everything will seem to run fine in the entire job, but at the end of your SAS log, you will see the following message:

```
ERROR: Errors printed on page ....
```

Like before, this is typically caused by one or more PROC FACTEX steps failing and the macro recovering and continuing. While the macro can sometimes suppress error messages, SAS still knows that a procedure tried to display an error message and displays an error at the end of the log. This error can be ignored.

[*]If this happens, please contact Technical Support. See page 25 for more information. I will see if I can make the macros better handle that problem in the next release. Send all the code necessary to reproduce what you have done.

Linear Models and Conjoint Analysis with Nonlinear Spline Transformations

Warren F. Kuhfeld

Mark Garratt

Abstract

Many common data analysis models are based on the general linear univariate model, including linear regression, analysis of variance, and conjoint analysis. This chapter discusses the general linear model in a framework that allows nonlinear transformations of the variables. We show how to evaluate the effect of each transformation. Applications to marketing research are presented.[*]

Why Use Nonlinear Transformations?

In marketing research, as in other areas of data analysis, relationships among variables are not always linear. Consider the problem of modeling product purchasing as a function of product price. Purchasing may decrease as price increases. For consumers who consider price to be an indication of quality, purchasing may increase as price increases but then start decreasing as the price gets too high. The number of purchases may be a discontinuous function of price with jumps at "round numbers" such as even dollar amounts. In any case, it is likely that purchasing behavior is not a linear function of price. Marketing researchers who model purchasing as a linear function of price may miss valuable nonlinear information in their data. A transformation regression model can be used to investigate the nonlinearities. The data analyst is not required to specify the form of the nonlinear function; the data suggest the function.

The primary purpose of this chapter is to suggest the use of linear regression models with nonlinear transformations of the variables—*transformation regression* models. It is common in marketing research to model nonlinearities by fitting a quadratic polynomial model. Polynomial models often have collinearity problems, but that can be overcome with orthogonal polynomials. The problem that polynomials cannot overcome is the fact that polynomial curves are rigid; they do not do a good job of locally fitting the data. Piecewise polynomials or *splines* are generalizations of polynomials that provide more flexibility than ordinary polynomials.

[*]This chapter is a revision of a paper that was presented to the American Marketing Association, Advanced Research Techniques Forum, June 14–17, 1992, Lake Tahoe, Nevada. The authors are: Warren F. Kuhfeld, Manager, Multivariate Models R&D, SAS Institute Inc., Cary NC 27513-2414. Mark Garratt was with Conway | Milliken & Associates, when this paper was presented and is now with In4mation Insights. Copies of this chapter (MR-2010J), sample code, and all of the macros are available on the Web `http://support.sas.com/resources/papers/tnote/tnote_marketresearch.html`. All plots in this chapter are produced using ODS Graphics. For help, please contact SAS Technical Support. See page 25 for more information.

Background and History

The foundation for our work can be found mostly in the psychometric literature. Some relevant references include: Kruskal & Shepard (1974); Young, de Leeuw, & Takane (1976); de Leeuw, Young, & Takane (1976); Perreault & Young (1980); Winsberg & Ramsay (1980); Young (1981); Gifi (1981, 1990); Coolen, van Rijckevorsel, & de Leeuw (1982); van Rijckevorsel (1982); van der Burg & de Leeuw (1983); de Leeuw (1986), and many others. The transformation regression problem has also received attention in the statistical literature (Breiman & Friedman 1985, Hastie & Tibshirani 1986) under the names *ACE* and *generalized additive models*.

Our work is characterized by the following statements:

- Transformation regression is an inferential statistical technique, not a purely descriptive technique.
- We prefer smooth nonlinear spline transformations over step-function transformations.
- Transformations are found that minimize a squared-error loss function.

Many of the models discussed in this chapter can be directly fit with some data manipulations and any multiple regression or canonical correlation software; some models require specialized software. Algorithms are given by Kuhfeld (1990), de Boor (1978), and in SAS/STAT documentation.

Next, we present notation and review some fundamentals of the general linear univariate model.

The General Linear Univariate Model

A general linear univariate model has the scalar form

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_m x_m + \epsilon$$

and the matrix form

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

The dependent variable $\mathbf{y}$ is an $(n \times 1)$ vector of observations; $\mathbf{y}$ has expected value $E(\mathbf{y}) = \mathbf{X}\boldsymbol{\beta}$ and expected variance $V(\mathbf{y}) = \sigma^2\mathbf{I}_n$. The vector $\boldsymbol{\epsilon} = \mathbf{y} - \mathbf{X}\boldsymbol{\beta}$ contains the unobservable deviations from the expected values. The assumptions on $\mathbf{y}$ imply $E(\epsilon) = \mathbf{0}$ and $V(\epsilon) = \sigma^2\mathbf{I}_n$. The columns of $\mathbf{X}$ are the independent variables; $\mathbf{X}$ is an $(n \times m)$ matrix of constants that are assumed to be known without appreciable error. The elements of the column vector $\boldsymbol{\beta}$ are the parameters. The objectives of a linear models analysis are to estimate the parameter vector $\boldsymbol{\beta}$, estimate interesting linear combinations of the elements of $\boldsymbol{\beta}$, and test hypotheses about the parameters $\boldsymbol{\beta}$ or linear combinations of $\boldsymbol{\beta}$.

We discuss fitting linear models with nonlinear spline transformations of the variables. Note that we do *not* discuss models that are nonlinear in the parameters such as

$$\begin{aligned} y &= e^{x\beta} + \epsilon \\ y &= \beta_0 x^{\beta_1} + \epsilon \\ y &= \frac{\beta_1 x_1 + \beta_2 x_1^2}{\beta_3 x_2 + \beta_4 x_2^2} + \epsilon \end{aligned}$$

Table 1
Cubic Polynomial
Spline Basis

1	-5	25	-125
1	-4	16	-64
1	-3	9	-27
1	-2	4	-8
1	-1	1	-1
1	0	0	0
1	1	1	1
1	2	4	8
1	3	9	27
1	4	16	64
1	5	25	125

Table 2
Cubic Polynomial
With Knots at −2, 0, 2

1	-5	25	-125	0	0	0
1	-4	16	-64	0	0	0
1	-3	9	-27	0	0	0
1	-2	4	-8	0	0	0
1	-1	1	-1	1	0	0
1	0	0	0	8	0	0
1	1	1	1	27	1	0
1	2	4	8	64	8	0
1	3	9	27	125	27	1
1	4	16	64	216	64	8
1	5	25	125	343	125	27

Table 3
Basis for a Discontinuous (at 0) Spline

1	-5	25	-125	0	0	0	0
1	-4	16	-64	0	0	0	0
1	-3	9	-27	0	0	0	0
1	-2	4	-8	0	0	0	0
1	-1	1	-1	0	0	0	0
1	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
1	2	4	8	1	2	4	8
1	3	9	27	1	3	9	27
1	4	16	64	1	4	16	64
1	5	25	125	1	5	25	125

Our nonlinear transformations are found within the framework of the general linear model.

There are numerous special cases of the general linear model that are of interest. When all of the columns of $\mathbf{y}$ and $\mathbf{X}$ are interval variables, the model is a multiple regression model. When all of the columns of $\mathbf{X}$ are indicator variables created from nominal variables, the model is a main-effects analysis of variance model, or a metric conjoint analysis model. The model

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \epsilon$$

is of special interest. It is a *linear* model because it is linear in the parameters, and it models y as a *nonlinear* function of x. It is a *cubic polynomial* regression model, which is a special case of a spline.

Polynomial Splines

Splines are curves that are typically required to be continuous and smooth. Splines are usually defined as piecewise polynomials of degree d whose function values and first $d-1$ derivatives agree at the points where they join. The abscissa values of the join points are called knots. The term spline is also used for polynomials (splines with no knots), and piecewise polynomials with more than one discontinuous derivative. Splines with more knots or more discontinuities fit the data better and use more degrees of freedom (*df*). Fewer knots and fewer discontinuities provide smoother splines that user fewer *df*. A spline of degree three is a cubic spline, degree two splines are quadratic splines, degree one splines are piecewise linear, and degree zero splines are step functions. Higher degrees are rarely used.

A simple special case of a spline is the line,

$$\beta_0 + \beta_1 x$$

from the simple regression model

$$y = \beta_0 + \beta_1 x + \epsilon$$

A line is continuous and completely smooth. However, there is little to be gained by thinking of a line as a spline. A special case of greater interest was mentioned previously. The polynomial

$$\beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$$

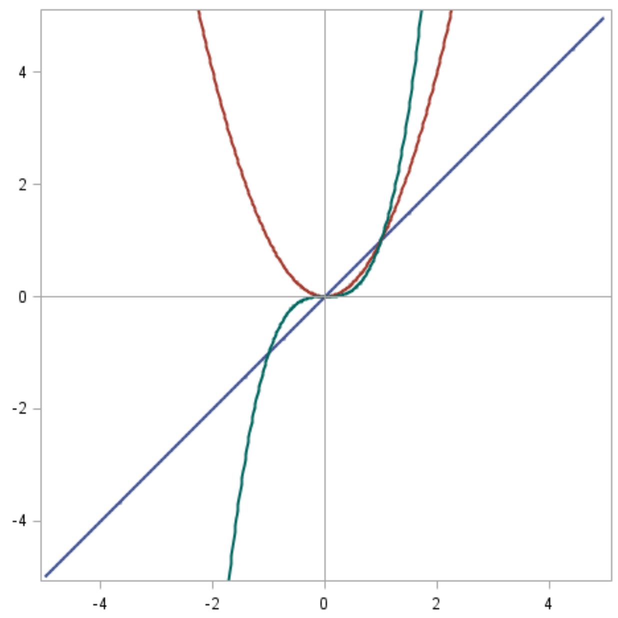


Figure 1. Linear, Quadratic, and Cubic Curves

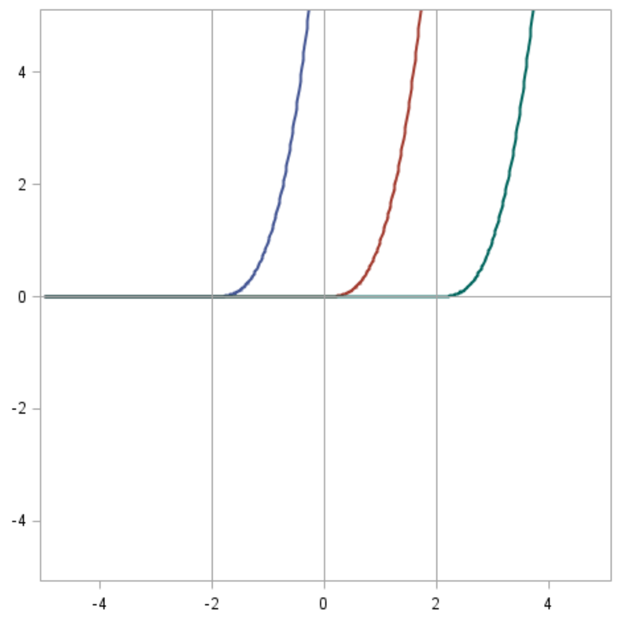


Figure 2. Curves For Knots at −2, 0, 2

from the linear model

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \epsilon$$

is a cubic spline with no knots. This equation models y as a *nonlinear* function of x, but does so with a *linear* regression model; y is a linear function of x, x^2, and x^3. Table 1 shows the $\mathbf{X}$ matrix, $(\mathbf{1}\ \ \mathbf{x}\ \ \mathbf{x}^2\ \ \mathbf{x}^3)$, for a cubic polynomial, where $x = -5, -4, ..., 5$. Figure 1 plots the polynomial terms (except the intercept). See Smith (1979) for an excellent introduction to splines.

Splines with Knots

Here is an example of a polynomial spline model with three knots at t_1, t_2, and t_3.

$$\begin{aligned} y \quad = \quad & \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \\ & \beta_4 (x > t_1)(x - t_1)^3 + \\ & \beta_5 (x > t_2)(x - t_2)^3 + \\ & \beta_6 (x > t_3)(x - t_3)^3 + \epsilon \end{aligned}$$

The Boolean expression $(x > t_1)$ is 1 if $x > t_1$, and 0 otherwise. The term

$$\beta_4 (x > t_1)(x - t_1)^3$$

is zero when $x \leq t_1$ and becomes nonzero, letting the curve change, as x becomes greater than knot t_1. This spline uses more *df* and is less smooth than the polynomial model

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \epsilon$$

Assume knots at −2, 0, and 2; the spline model is:

$$y \quad = \quad \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 +$$

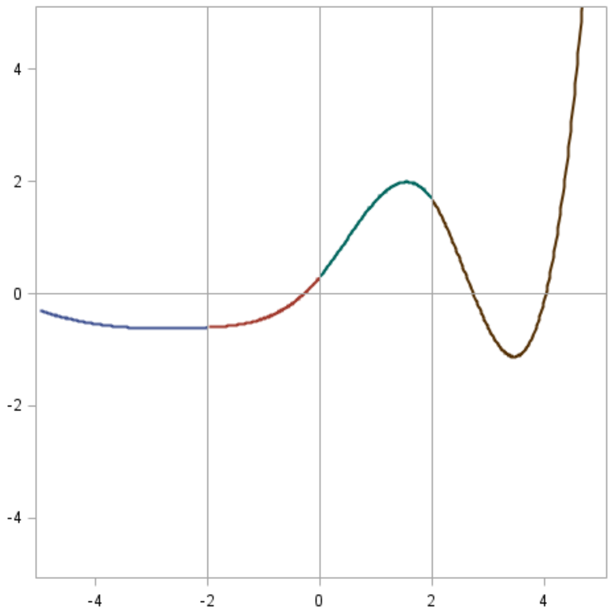


Figure 3. A Spline Curve With Knots at −2, 0, 2

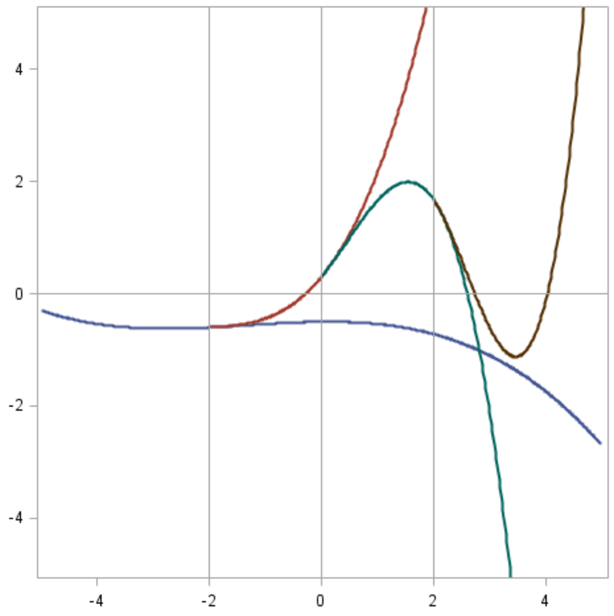


Figure 4. The Components of the Spline

$$\begin{aligned}&\beta_4(x > -2)(x - -2)^3 + \\ &\beta_5(x > \quad 0)(x - \quad 0)^3 + \\ &\beta_6(x > \quad 2)(x - \quad 2)^3 + \epsilon\end{aligned}$$

Table 2 shows an **X** matrix for this model, Figure 1 plots the polynomial terms, and Figure 2 plots the knot terms.

The β_0, $\beta_1 x$, $\beta_2 x^2$, and $\beta_3 x^3$ terms contribute to the overall shape of the curve. The

$$\beta_4(x > -2)(x - -2)^3$$

term has no effect on the curve before $x = -2$, and allows the curve to change at $x = -2$. The $\beta_4(x > -2)(x - -2)^3$ term is exactly zero at $x = -2$ and increases as x becomes greater than -2. The $\beta_4(x > -2)(x - -2)^3$ term contributes to the shape of the curve even beyond the next knot at $x = 0$, but at $x = 0$,

$$\beta_5(x > 0)(x - 0)^3$$

allows the curve to change again. Finally, the last term

$$\beta_6(x > 2)(x - 2)^3$$

allows one more change. For example, consider the curve in Figure 3. It is the spline

$$\begin{aligned}y \quad = \quad & -0.5 + 0.01x + -0.04x^2 + -0.01x^3 + \\ & 0.1(x > -2)(x - -2)^3 + \\ & -0.5(x > \quad 0)(x - \quad 0)^3 + \\ & 1.5(x > \quad 2)(x - \quad 2)^3\end{aligned}$$

It is constructed from the curves in Figure 4. At $x = -2.0$ there is a branch;

$$y = -0.5 + 0.01x + -0.04x^2 + -0.01x^3$$

continues over and down while the curve of interest,

$$\begin{aligned} y &= -0.5 + 0.01x + -0.04x^2 + -0.01x^3 + \\ &\quad 0.1(x > -2)(x - -2)^3 \end{aligned}$$

starts heading upwards. At $x = 0$, the addition of

$$-0.5(x > 0)(x - 0)^3$$

slows the ascent until the curve starts decreasing again. Finally, the addition of

$$1.5(x > 2)(x - 2)^3$$

produces the final change. Notice that the curves do not immediately diverge at the knots. The function and its first two derivatives are continuous, so the function is smooth everywhere.

Derivatives of a Polynomial Spline

The next equations show a cubic spline model with a knot at t_1 and its first three derivatives with respect to x.

$$\begin{aligned} y &= \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \\ &\quad \beta_4 (x > t_1)(x - t_1)^3 + \epsilon \end{aligned}$$

$$\begin{aligned} \frac{dy}{dx} &= \beta_1 + 2\beta_2 x + 3\beta_3 x^2 + \\ &\quad 3\beta_4 (x > t_1)(x - t_1)^2 \end{aligned}$$

$$\begin{aligned} \frac{d^2y}{dx^2} &= 2\beta_2 + 6\beta_3 x + \\ &\quad 6\beta_4 (x > t_1)(x - t_1) \end{aligned}$$

$$\frac{d^3y}{dx^3} = 6\beta_3 + 6\beta_4 (x > t_1)$$

The first two derivatives are continuous functions of x at the knots. This is what gives the spline function its smoothness at the knots. In the vicinity of the knots, the curve is continuous, the slope of the curve is a continuous function, and the rate of change of the slope function is a continuous function. The third derivative is discontinuous at the knots. It is the horizontal line $6\beta_3$ when $x \leq t_1$ and jumps to the horizontal line $6\beta_3 + 6\beta_4$ when $x > t_1$. In other words, the cubic part of the curve changes at the knots, but the linear and quadratic parts do not change.

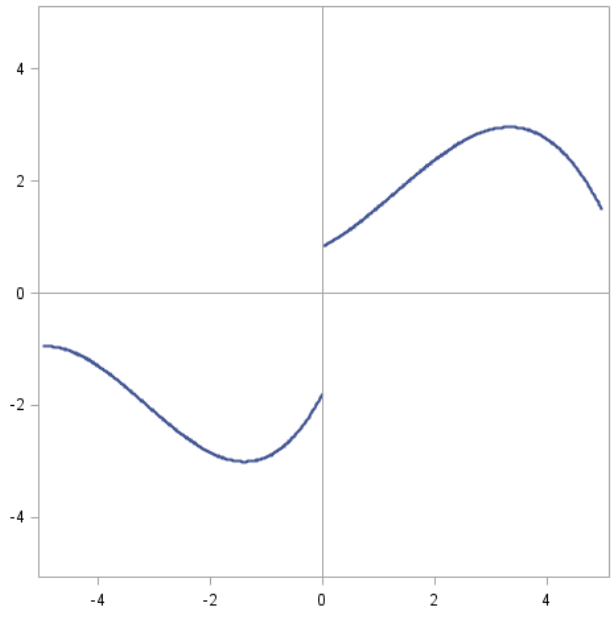


Figure 5. A Discontinuous Spline Function

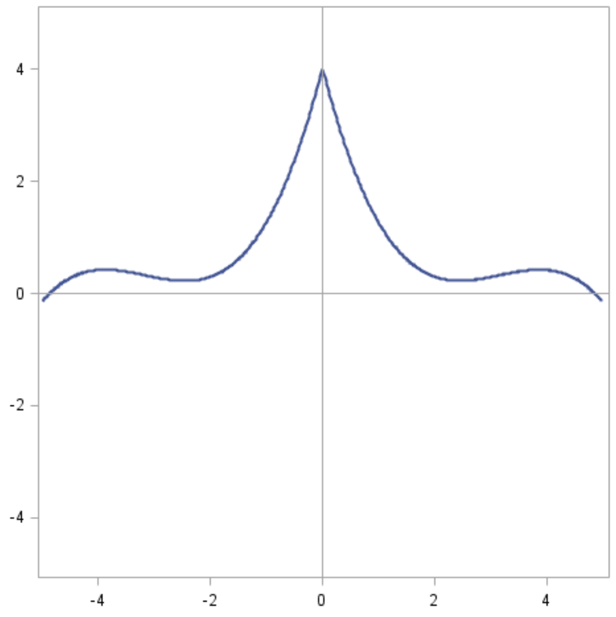


Figure 6. A Spline With a Discontinuous Slope

Discontinuous Spline Functions

Here is an example of a spline model that is discontinuous at $x = t_1$.

$$\begin{aligned} y &= \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \\ &\quad \beta_4 (x > t_1) + \\ &\quad \beta_5 (x > t_1)(x - t_1) + \\ &\quad \beta_6 (x > t_1)(x - t_1)^2 + \\ &\quad \beta_7 (x > t_1)(x - t_1)^3 + \epsilon \end{aligned}$$

Figure 5 shows an example, and Table 3 shows a design matrix for this model with $t_1 = 0$. The fifth column is a binary (zero/one) vector that allows the discontinuity. It is a change in the intercept parameter. Note that the sixth through eighth columns are necessary if the spine is to consist of two independent polynomials. Without them, there is a jump at $t_1 = 0$, but both curves are based on the same polynomial. For $x \leq t_1$, the spline is

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \epsilon$$

and for $x > t_1$, the spline is

$$\begin{aligned} y &= \beta_0 + \beta_4 + \\ &\quad \beta_1 x + \beta_5 (x - t_1) + \\ &\quad \beta_2 x^2 + \beta_6 (x - t_1)^2 + \\ &\quad \beta_3 x^3 + \beta_7 (x - t_1)^3 + \epsilon \end{aligned}$$

The discontinuities are as follows:

$$\beta_7 (x > t_1)(x - t_1)^3$$

specifies a discontinuity in the third derivative of the spline function at t_1,

$$\beta_6 (x > t_1)(x - t_1)^2$$

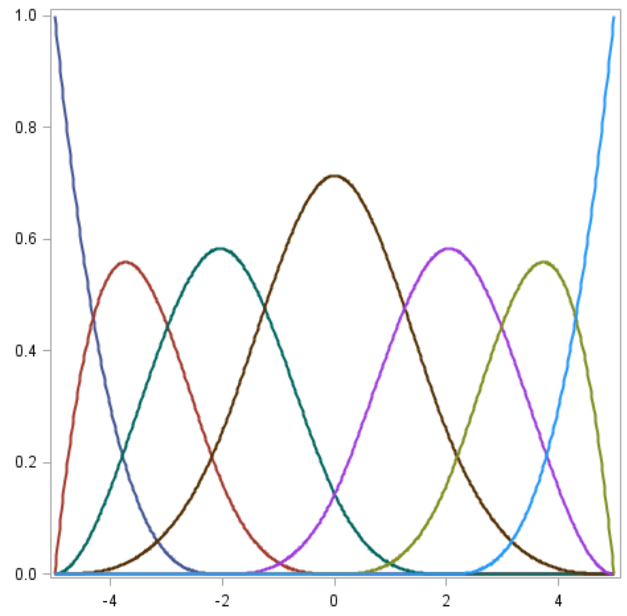


Figure 7. B-Splines With Knots at −2, 0, 2

Table 4
Cubic B-Spline With Knots at −2, 0, 2

1.00	0.00	0.00	0.00	0.00	0.00	0.00
0.30	0.54	0.15	0.01	0.00	0.00	0.00
0.04	0.45	0.44	0.08	0.00	0.00	0.00
0.00	0.16	0.58	0.26	0.00	0.00	0.00
0.00	0.02	0.41	0.55	0.02	0.00	0.00
0.00	0.00	0.14	0.71	0.14	0.00	0.00
0.00	0.00	0.02	0.55	0.41	0.02	0.00
0.00	0.00	0.00	0.26	0.58	0.16	0.00
0.00	0.00	0.00	0.08	0.44	0.45	0.04
0.00	0.00	0.00	0.01	0.15	0.54	0.30
0.00	0.00	0.00	0.00	0.00	0.00	1.00

specifies a discontinuity in the second derivative at t_1,

$$\beta_5(x > t_1)(x - t_1)$$

specifies a discontinuity in the derivative at t_1, and

$$\beta_4(x > t_1)$$

specifies a discontinuity in the function at t_1. The function consists of two separate polynomial curves, one for $-\infty < x \leq t_1$ and the other for $t_1 < x < \infty$. This kind of spline can be used to model a discontinuity in price.

Here is an example of a spline model that is continuous at $x = t_1$ but does not have $d - 1$ continuous derivatives.

$$\begin{aligned} y &= \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \\ &\quad \beta_4(x > t_1)(x - t_1) + \\ &\quad \beta_5(x > t_1)(x - t_1)^2 + \\ &\quad \beta_6(x > t_1)(x - t_1)^3 + \epsilon \end{aligned}$$

$$\begin{aligned} \frac{dy}{dx} &= \beta_1 + 2\beta_2 x + 3\beta_3 x^2 + \\ &\quad \beta_4(x > t_1) + \\ &\quad 2\beta_5(x > t_1)(x - t_1) + \\ &\quad 3\beta_6(x > t_1)(x - t_1)^2 \end{aligned}$$

Since the first derivative is not continuous at $t_1 = x$, the slope of the spline is not continuous at $t_1 = x$. Figure 6 contains an example with $t_1 = 0$. Notice that the slope of the curve is indeterminate at zero.

Table 5

Polynomial and B-Spline Eigenvalues

B-Spline Basis

Eigenvalue	Proportion	Cumulative
0.107872	0.358718	0.35872
0.096710	0.321599	0.68032
0.046290	0.153933	0.83425
0.030391	0.101062	0.93531
0.012894	0.042878	0.97819
0.006559	0.021810	1.00000

Polynomial Spline Basis

Eigenvalue	Proportion	Cumulative
10749.8	0.941206	0.94121
631.8	0.055317	0.99652
37.7	0.003300	0.99982
1.7	0.000148	0.99997
0.3	0.000029	1.00000
0.0	0.000000	1.00000

Monotone Splines and B-Splines

An increasing *monotone spline* never decreases; its slope is always positive or zero. Decreasing monotone splines, with slopes that are always negative or zero, are also possible. Monotone splines cannot be conveniently created from polynomial splines. A different basis, the *B-spline* basis, is used instead. Polynomial splines provide a convenient way to describe splines, but B-splines provide a better way to fit spline models.

The columns of the B-spline basis are easily constructed with a recursive algorithm (de Boor 1978, pages 134–135). A basis for a vector space is a linearly independent set of vectors; every vector in the space has a unique representation as a linear combination of a given basis. Table 4 shows the B-spline $\mathbf{X}$ matrix for a model with knots at -2, 0, and 2. Figure 7 shows the B-spline curves. The columns of the matrix in Table 4 can all be constructed by taking linear combinations of the columns of the polynomial spline in Table 2. Both matrices form a basis for the same vector space.

The numbers in the B-spline basis are all between zero and one, and the marginal sums across columns are all ones. The matrix has a diagonally banded structure, such that the band moves one position to the right at each knot. The matrix has many more zeros than the matrix of polynomials and much smaller numbers. The columns of the matrix are not orthogonal like a matrix of orthogonal polynomials, but collinearity is not a problem with the B-spline basis like it is with a polynomial spline. The B-spline basis is very stable numerically.

To illustrate, 1000 evenly spaced observations were generated over the range -5 to 5. Then a B-spline basis and polynomial spline basis were constructed with knots at -2, 0, and 2. The eigenvalues for the centered $\mathbf{X}'\mathbf{X}$ matrices, excluding the last structural zero eigenvalue, are given in Table 5. In the polynomial splines, the first two components already account for more than 99% of the variation of the points. In the B-splines, the cumulative proportion does not pass 99% until the last term. The eigenvalues show that the B-spline basis is better conditioned than the piecewise polynomial basis. B-splines are used instead of orthogonal polynomials or Box-Cox transformations because B-splines allow knots and more general curves. B-splines also allow monotonicity constraints.

A transformation of x is monotonically increasing if the coefficients that are used to combine the columns of the B-spline basis are monotonically increasing. Models with splines can be fit directly using ordinary least squares (OLS). OLS does not work for monotone splines because OLS has no method of ensuring monotonicity in the coefficients. When there are monotonicity constraints, an alternating least square (ALS) algorithm is used. Both OLS and ALS attempt to minimize a squared error loss function. See Kuhfeld (1990) for a description of the iterative algorithm that fits monotone splines. See Ramsay (1988) for some applications and a different approach to ensuring monotonicity.

Transformation Regression

If the dependent variable is not transformed and if there are no monotonicity constraints on the independent variable transformations, the transformation regression model is the same as the OLS regression model. When only the independent variables are transformed, the transformation regression model is nothing more than a different rendition of an OLS regression. All of the spline models presented up to this point can be reformulated as

$$y = \beta_0 + \Phi(x) + \epsilon$$

The nonlinear transformation of x is $\Phi(x)$; it is solved for by fitting a spline model such as

$$\begin{aligned} y &= \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \\ &\quad \beta_4 (x > t_1)(x - t_1)^3 + \\ &\quad \beta_5 (x > t_2)(x - t_2)^3 + \\ &\quad \beta_6 (x > t_3)(x - t_3)^3 + \epsilon \end{aligned}$$

where

$$\begin{aligned} \Phi(x) &= \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \\ &\quad \beta_4 (x > t_1)(x - t_1)^3 + \\ &\quad \beta_5 (x > t_2)(x - t_2)^3 + \\ &\quad \beta_6 (x > t_3)(x - t_3)^3 \end{aligned}$$

Consider a model with two polynomials:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_1^3 + \beta_4 x_2 + \beta_5 x_2^2 + \beta_6 x_2^3 + \epsilon$$

It is the same as a transformation regression model

$$y = \beta_0 + \Phi_1(x_1) + \Phi_2(x_2) + \epsilon$$

where $\Phi_\bullet(\bullet)$ designates cubic spline transformations with no knots. The actual transformations in this case are

$$\widehat{\Phi}_1(x_1) = \widehat{\beta}_1 x_1 + \widehat{\beta}_2 x_1^2 + \widehat{\beta}_3 x_1^3$$

and

$$\widehat{\Phi}_2(x_2) = \widehat{\beta}_4 x_2 + \widehat{\beta}_5 x_2^2 + \widehat{\beta}_6 x_2^3$$

There are six model *df*. The test for the effect of the transformation $\Phi_1(x_1)$ is the test of the linear hypothesis $\beta_1 = \beta_2 = \beta_3 = 0$, and the $\Phi_2(x_2)$ transformation test is a test that $\beta_4 = \beta_5 = \beta_6 = 0$. Both tests are *F*-tests with three numerator *df*. When there are monotone transformations, constrained least-squares estimates of the parameters are obtained.

Degrees of Freedom

In an ordinary general linear model, there is one parameter for each independent variable. In the transformation regression model, many of the variables are used internally in the bases for the transformations. Each linearly independent basis column has one parameter and one model *df*. If a variable is not transformed, it has one parameter. Nominal classification variables with c categories have $c - 1$ parameters. For degree d splines with k knots and $d - 1$ continuous derivatives, there are $d + k$ parameters.

When there are monotonicity constraints, counting the number of scoring parameters is less precise. One way of handling a monotone spline transformation is to treat it as if it were simply a spline transformation with $d + k$ parameters. However, there are typically fewer than $d + k$ *unique* parameter estimates since some of those $d + k$ scoring parameter estimates may be tied to impose the order constraints. Imposing ties is equivalent to fitting a model with fewer parameters. So, there are two available scoring parameter counts: $d + k$ and a potentially smaller number that is determined during the analysis. Using $d + k$ as the model *df* is *conservative* since the scoring parameter estimates are restricted. Using the smaller count is too *liberal* since the data and the model together are being used to determine the number of parameters. Our solution is to report tests using both liberal and conservative *df* to provide lower and upper bounds on the "true" p-values.

Dependent Variable Transformations

When a dependent variable is transformed, the problem becomes multivariate:

$$\Phi_0(y) = \beta_0 + \Phi_1(x_1) + \Phi_2(x_2) + \epsilon$$

Hypothesis tests are performed in the context of a multivariate linear model, with the number of dependent variables equal to the number of scoring parameters for the dependent variable transformation. Multivariate normality is assumed even though it is known that the assumption is *always* violated. This is one reason that all hypothesis tests in the presence of a dependent variable transformation should be considered approximate at best.

For the transformation regression model, we redefine three of the usual multivariate test statistics: Pillai's Trace, Wilks' Lambda, and the Hotelling-Lawley Trace. These statistics are normally computed using all of the squared canonical correlations, which are the eigenvalues of the matrix $\mathbf{H}(\mathbf{H} + \mathbf{E})^{-1}$. Here, there is only one linear combination (the transformation) and hence only one squared canonical correlation of interest, which is equal to the R^2. We use R^2 for the first eigenvalue; all other eigenvalues are set to zero since only one linear combination is used. Degrees of freedom are computed assuming that all linear combinations contribute to the Lambda and Trace statistics, so the F-tests for those statistics are conservative. In practice, the adjusted Pillai's Trace is very conservative—perhaps too conservative to be useful. Wilks' Lambda is less conservative, and the Hotelling-Lawley Trace seems to be the least conservative.

It may seem that the Roy's Greatest Root statistic, which always uses only the largest squared canonical correlation, is the only statistic of interest. Unfortunately, Roy's Greatest Root is very liberal and only provides a lower bound on the p-value. The p-values for the liberal and conservative statistics are used together to provide approximate lower and upper bounds on p.

Scales of Measurement

Early work in scaling, such as Young, de Leeuw, & Takane (1976), and Perreault & Young (1980) was concerned with fitting models with mixed nominal, ordinal, and interval scale of measurement variables. Nominal variables were optimally scored using Fisher's (1938) optimal scoring algorithm. Ordinal variables were optimally scored using the Kruskal and Shepard (1974) monotone regression algorithm. Interval and ratio scale of measurement variables were left alone nonlinearly transformed with a polynomial transformation.

In the transformation regression setting, the Fisher optimal scoring approach is equivalent to using an indicator variable representation, as long as the correct *df* are used. The optimal scores are category means. Introducing optimal scaling for nominal variables does not lead to any increased capability in the regression model.

For ordinal variables, we believe the Kruskal and Shepard monotone regression algorithm should be reserved for the situation when a variable has only a few categories, say five or fewer. When there are more levels, a monotone spline is preferred because it uses fewer model *df* and because it is less likely to capitalize on chance.

Interval and ratio scale of measurement variables can be left alone or nonlinearly transformed with splines or monotone splines. When the true model has a nonlinear function, say

$$y = \beta_0 + \beta_1 log(x) + \epsilon$$

or

$$y = \beta_0 + \beta_1/x + \epsilon$$

the transformation regression model

$$y = \beta_0 + \Phi(x) + \epsilon$$

can be used to hunt for parametric transformations. Plots of $\widehat{\Phi}(x)$ may suggest log or reciprocal transformations.

Conjoint Analysis

Green & Srinivasan (1990) discuss some of the problems that can be handled with a transformation regression model, particularly the problem of degrees of freedom. Consider a conjoint analysis design where a factor with $c > 3$ levels has an inherent ordering. By finding a quadratic monotone spline transformation with no knots, that variable will use only two *df* instead of the larger $c - 1$. The model *df* in a spline transformation model are determined by the data analyst, not by the number of categories in the variables. Furthermore, a "*quasi-metric*" conjoint analysis can be performed by finding a monotone spline transformation of the dependent variable. This model has fewer restrictions than a metric analysis, but will still typically have error *df*, unlike the nonmetric analysis.

Curve Fitting Applications

With a simple regression model, you can fit a line through a $y \times x$ scatter plot. With a transformation regression model, you can fit a curve through the scatter plot. The y-axis coordinates of the curve are

$$\widehat{y} = \widehat{\beta}_0 + \widehat{\Phi}(x)$$

from the model

$$y = \beta_0 + \Phi(x) + \epsilon$$

With more than one group of observations and a multiple regression model, you can fit multiple lines, lines with the same slope but different intercepts, and lines with common intercepts but different slopes. With the transformation regression model, you can fit multiple curves through a scatter plot. The curves can be monotone or not, constrained to be parallel, or constrained to have the same intercept. Consider the problem of modeling the number of product purchases as a function of price. Separate curves can be simultaneously fit for two groups who may behave differently, for example those who are making a planned purchase and those who are buying impulsively. Later in this chapter, there is an example of plotting brand by price interactions.

Figure 8 contains an artificial example of two separate spline functions; the shapes of the two curves are independent of each other, and $R^2 = 0.87$. In Figure 9, the splines are constrained to be parallel, and $R^2 = 0.72$. The parallel curve model is more restrictive and fits the data less well than the unconstrained model. In Figure 8, each curve follows its swarm of data. In Figure 9, the curves find paths through the data that are best on the average considering both swarms together. In the vicinity of $x = -2$, the top curve is high and the bottom curve is low. In the vicinity of $x = 1$, the top curve is low and the bottom curve is high.

Figure 10 contains the same data and two monotonic spline functions; the shapes of the two curves are independent of each other, and $R^2 = 0.71$. The top curve is monotonically decreasing, whereas the bottom curve is monotonically increasing. The curves in Figure 10 flatten where there is nonmonotonicity in Figure 8.

Parallel curves are very easy to model. If there are two groups and the variable g is a binary variable indicating group membership, fit the model

$$y = \beta_0 + \beta_1 g + \Phi_1(x) + \epsilon$$

where $\Phi_1(x)$ is a linear, spline, or monotone spline transformation. Plot $\widehat{y}$ as a function of x to see the two curves. Separate curves are almost as easy; the model is

$$y = \beta_0 + \beta_1 g + \Phi_1(x \times (1 - g)) + \Phi_2(x \times g) + \epsilon$$

When $x \times (1 - g)$ is zero, $x \times g$ is x, and vice versa.

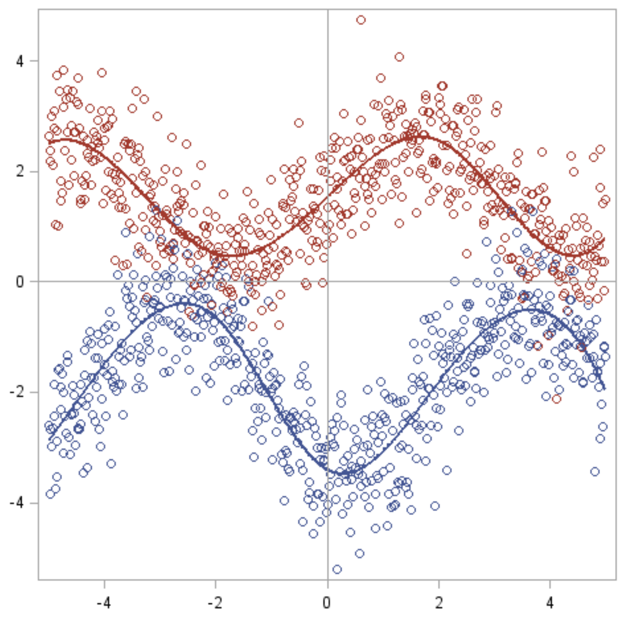


Figure 8. Separate Spline Functions, Two Groups

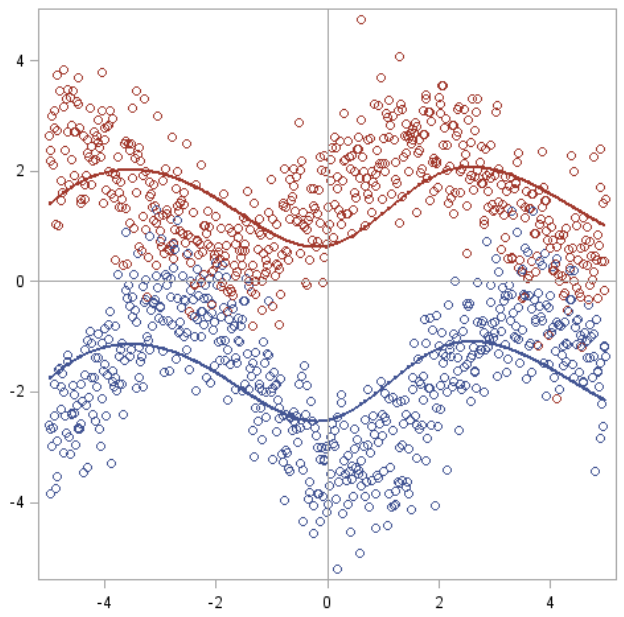


Figure 9. Parallel Spline Functions, Two Groups

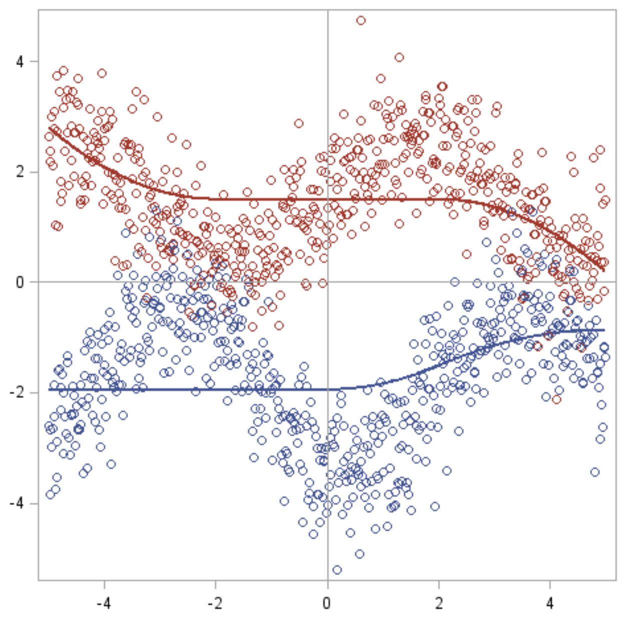


Figure 10. Monotone Spline Functions, Two Groups

Spline Functions of Price

This section illustrates splines with an artificial data set. Imagine that subjects were asked to rate their interest in purchasing various types of spaghetti sauces on a one to nine scale, where nine indicated definitely will buy and one indicated definitely will *not* buy. Prices were chosen from typical retail trade prices, such as $1.49, $1.99, $2.49, and $2.99; and one penny more than a typical price, $1.00, $1.50, $2.00, and $2.50. Between each "round" number price, such as $1.00, and each typical price, such as $1.49, three additional prices were chosen, such as $1.15, $1.25, and $1.35. The goal is to allow a model with a separate spline for each of the four ranges: $1.00 — $1.49, $1.50 — $1.99, $2.00 — $2.49, and $2.50 — $2.99. For each range, a spline with zero or one knot can be fit.

One rating for each price was constructed and various models were fit to the data. Figures 11 through 18 contain results. For each figure, the number of model *df* are displayed. One additional *df* for the intercept is also used. The SAS/STAT procedure TRANSREG was used to fit all of the models in this chapter.

Figure 11 shows the linear fit, Figure 12 uses a quadratic polynomial, and Figure 13 uses a cubic polynomial. The curve in Figure 13 has a slight nonmonotonicity in the tail, and since it is a polynomial, it is rigid and cannot locally fit the data values.

Figure 14 shows a monotone spline. It closely follows the data and never increases. A range for the model *df* is specified; the larger value is a conservative count and the smaller value is a liberal count.

The curves in Figures 12 through 14 are all continuous and smooth. These curves do a good job of following the data, but inspection of the data suggests that a different model may be more appropriate. There is a large drop in purchase interest when price increases from $1.49 to $1.50, a smaller drop between $1.99 and $2.00, and a still smaller drop between $2.49 and $2.50.

In Figure 15, a separate quadratic polynomial is fit for each of the four price ranges: $1.00 — $1.49, $1.50 — $1.99, $2.00 — $2.49, and $2.50 — $2.99. The functions are connected. The function over the range $1.00 — $1.49 is nearly flat; there is high purchase interest for all of these prices. Over the range $1.50 — $1.99, purchase interest drops more rapidly with a slight leveling in the low end; the slope decreases as the function increases. Over the range $2.00 — $2.49, purchase interest drops less rapidly; the slope increases as the function increases. Over the range $2.50 — $2.99, the function is nearly flat. At $1.99, $2.49, and $2.99 there is a slight increase in purchase interest.

In Figure 16, there is a knot in the middle of each range. This gives the spline more freedom to follow the data. Figure 17 uses the same model as Figure 16, but monotonicity is imposed. When monotonicity is imposed the curves touch fewer of the data values, passing in between the nonmonotonic points. In Figure 18, the means for each price are plotted and connected. This analysis uses the most model *df* and is the least smooth of the plots.

Benefits of Splines

In marketing research and conjoint analysis, the use of spline models can have several benefits. Whenever a factor has three or more levels and an inherent ordering of the levels, that factor can be modeled as a quadratic monotone spline. The *df* used by the variable is controlled at data analysis time; it is not simply the number of categories minus one. When the alternative is a model in which a factor is designated as nominal, splines can be used to fit a more restrictive model with fewer model *df*. Since the spline model has fewer *df*, it should yield more reproducible results.

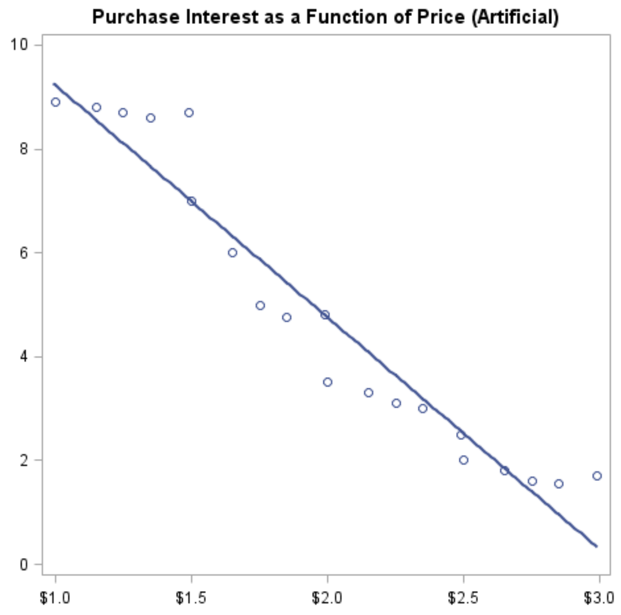


Figure 11. Linear Function, 1 df

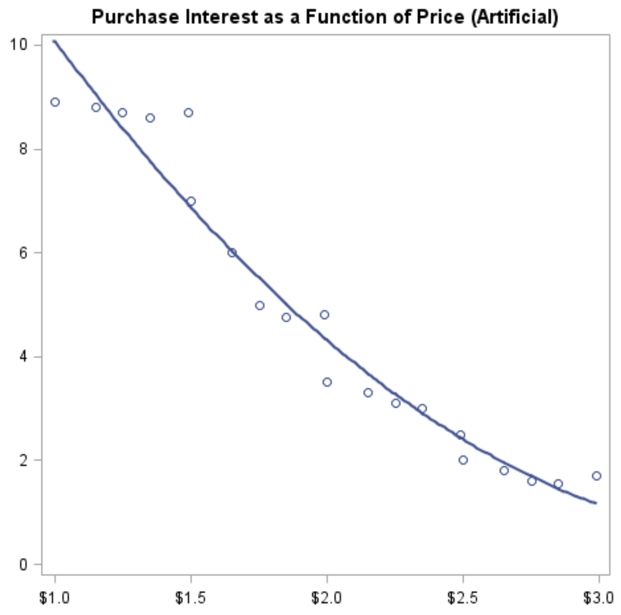


Figure 12. Quadratic Function, 2 df

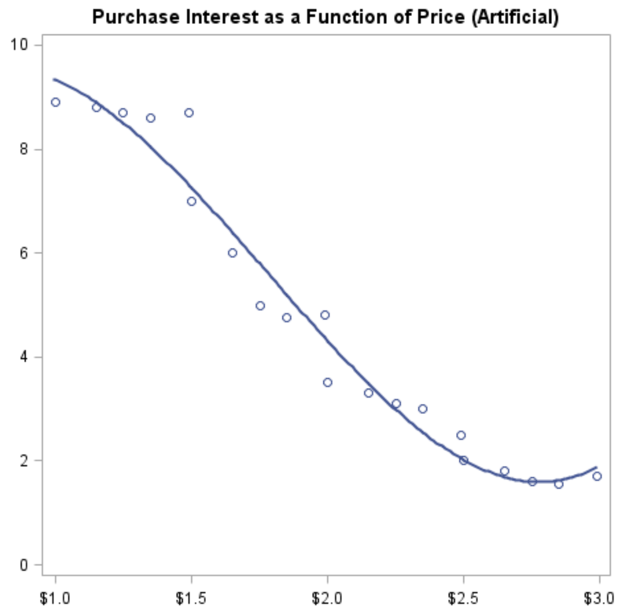


Figure 13. Cubic Function, 3 df

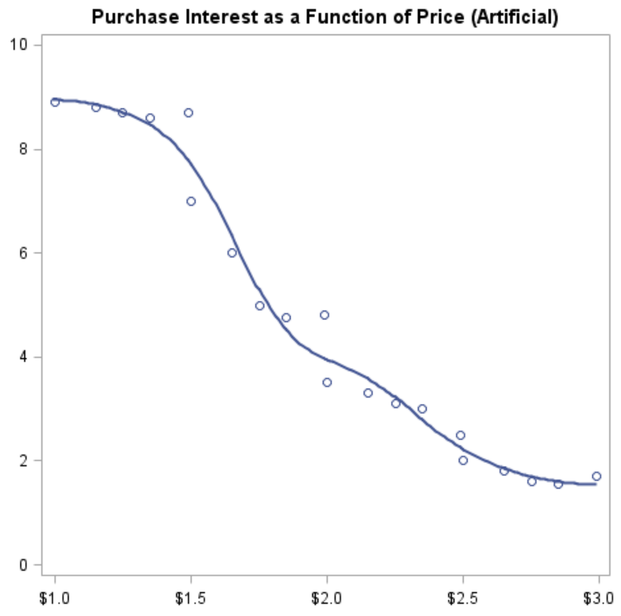


Figure 14. Monotone Function, 5–7 df

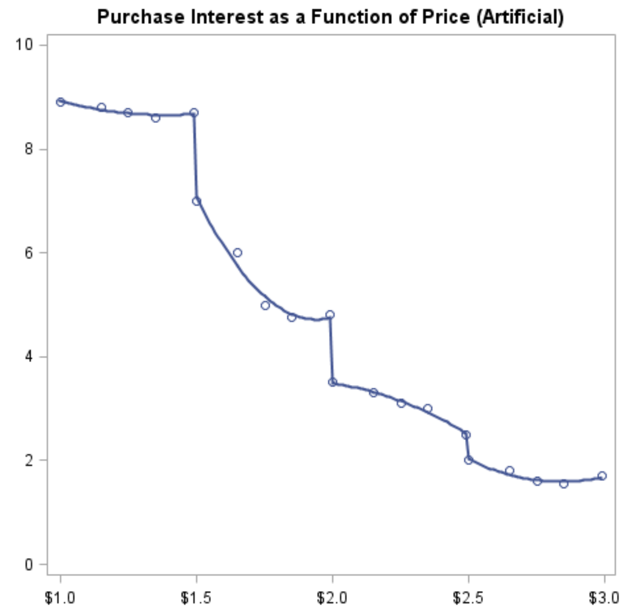


Figure 15. Discontinuous Polynomial, 11 df

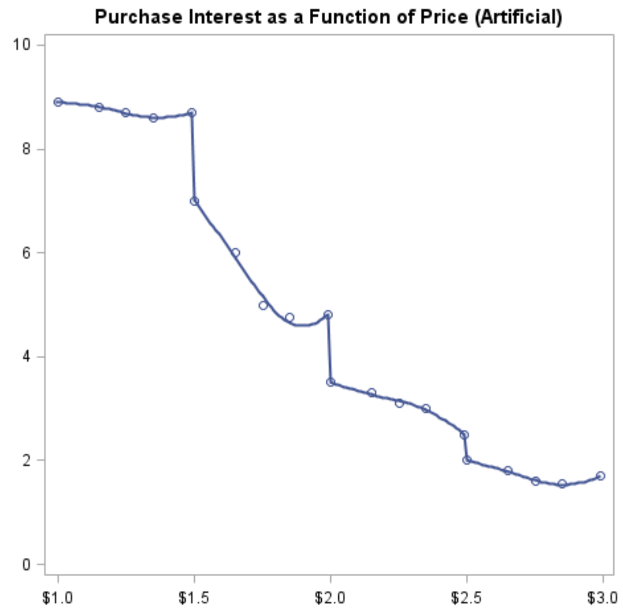


Figure 16. Discontinuous Spline Function, 15 df

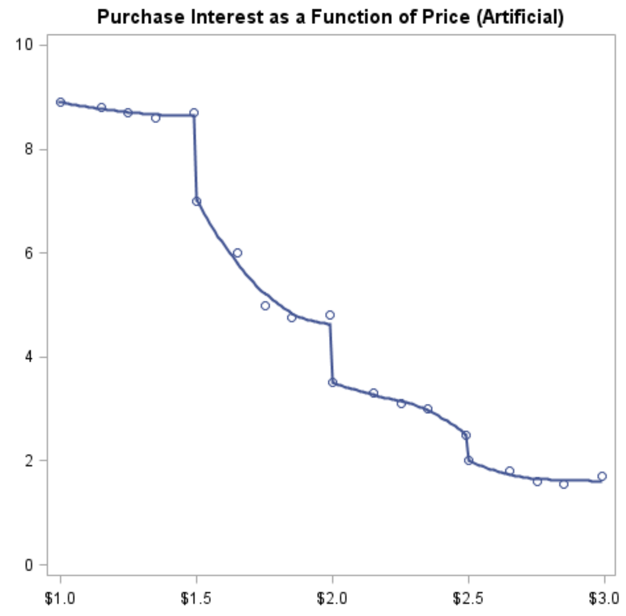


Figure 17. Discontinuous Monotone Spline, 12–15 df

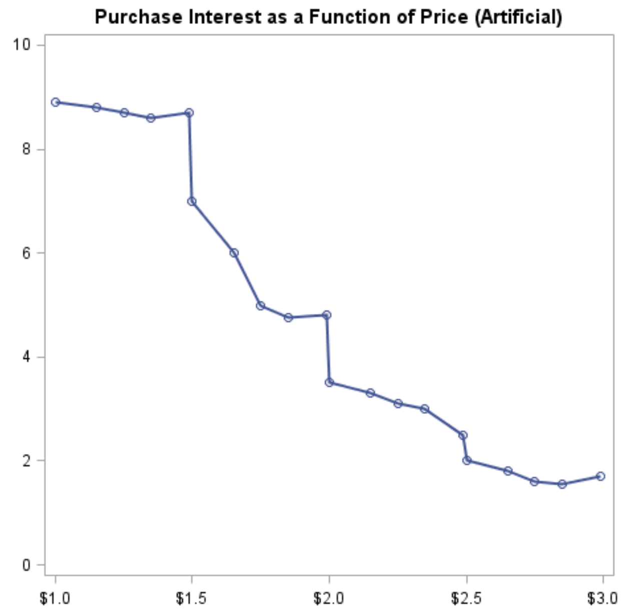


Figure 18. Cell Means, 19 df

The opposite alternative is also important. Consider a variable with many values, like price in some examples. Instead of using a restrictive single *df* linear model, splines can be used to fit a more general model with more *df*. The more general model may show information in the data that is not apparent from the ordinary linear model. This can be a benefit in conjoint analyses that focus on price, in the analysis of scanner data, and in survey research. Splines give you the ability to examine the nonlinearities that may be very important in predicting consumer behavior.

Fitting quadratic and cubic polynomial models is common in marketing research. Splines extend that capability by adding the possibility of knots and hence more general curves. Spline curves can also be restricted to be monotone. Monotone splines are less restrictive than a line and more restrictive than splines that can have both positive and negative slopes. You are no longer restricted to fitting just a line, polynomial, or a step function. Splines give you possibilities in between.

Conclusions

Splines allow you to fit curves to your data. Splines may not revolutionize the way you analyze data, but they will provide you with some new tools for your data analysis toolbox. These new tools allow you to try new methods for solving old problems and tackle new problems that could not be adequately solved with your old tools. We hope you will find these tools useful, and we hope that they will help you to better understand your marketing data.

Graphical Scatter Plots of Labeled Points

Warren F. Kuhfeld

Abstract

The `%PlotIt` (PLOT ITeratively) macro creates graphical scatter plots of labeled points. It is designed to make it easy to display raw data, regressions, and the results of many other data analyses. You can draw curves, vectors, and circles, and you can control the colors, sizes, fonts, and general appearance of the plots. The `%PlotIt` macro is a part of the SAS autocall library.[*]

Introduction

SAS has provided software for producing scatter plots for many years (for example, the PLOT and GPLOT procedures). For many types of data analyses, it is useful to have each point in the plot labeled with the value of a variable. However, before the creation of the `%PlotIt` macro, there was not a satisfactory way to do this. PROC GPLOT produces graphical scatter plots. Combined with the Annotate facility, it allows long point labels, but it does not provide any way to optimally position them. The PLOT procedure can optimally position long point labels in the scatter plot, however PROC PLOT cannot create a graphical scatter plot. The PROC PLOT label-placement algorithm was developed by Kuhfeld (1991), and the PROC PLOT options are documented in Base SAS documentation.

The macro, `%PlotIt` (PLOT ITeratively), creates graphical scatter plots of labeled points. It can fit curves, draw vectors, and draw circles. It has many options, but only a small number are needed for many types of plots. The `%PlotIt` macro uses DATA steps and multiple procedures, including PLOT and GANNO. The `%PlotIt` macro is almost 6000 lines long, so it is not displayed here. It is fully documented starting on page 1178 and in the header comments. This article illustrates through examples some of the main features of the `%PlotIt` macro.

For many years, the `%PlotIt` macro provided the only convenient way to get graphical scatter plots of labeled points. With SAS Version 9.2, however, there are other and in many cases better options for making these plots. Now, ODS Graphics automatically does much of what the `%PlotIt` macro was originally designed for, and usually, ODS Graphics does it better and more conveniently. Comparisons between the `%PlotIt` macro and ODS Graphics are as follows. The `%PlotIt` macro has a much more

[*]Copies of this chapter (MR-2010K), the other chapters, sample code, and all of the macros are available on the Web `http://support.sas.com/resources/papers/tnote/tnote_marketresearch.html`. This chapter originally appeared in the SAS Journal **Observations**, Fourth Quarter, 1994, pages 23–37. This version of the chapter has been updated for SAS Version 9.2. For help, please contact SAS Technical Support. See page 25 for more information.

sophisticated algorithm for placing labels and avoiding label collisions. The `%PlotIt` macro has a few other capabilities that are not in ODS Graphics (e.g. more sophisticated color ramps or "painting" for some applications). ODS Graphics is superior to the `%PlotIt` macro in virtually every other way, and ODS Graphics has *many* capabilities that the `%PlotIt` macro does not have. The algorithm that ODS Graphics has for label placement in 9.2, while clearly nonoptimal, is good enough for many analyses. This paper illustrates the traditional uses of the `%PlotIt` macro and points out cases where ODS Graphics is a suitable alternative. There is an appendix at the end of this chapter showing many of the same plots that the `%PlotIt` macro made, but this time made with ODS Graphics.

An Overview of the %PlotIt Macro

The `%PlotIt` macro performs the following steps.

1. It reads an input data set and preprocesses it. The preprocessed data set contains information such as the axis variables, the point-symbol and point-label variables, and symbol and label types, sizes, fonts, and colors. The nature of the preprocessing depends on the type of data analysis that generated the input data set. For example, if the option DATATYPE=MDPREF was specified with an input data set created by PROC PRINQUAL for a multidimensional preference analysis, then the `%PlotIt` macro creates by default blue points for `_TYPE_ = 'SCORE'` observations and red vectors for `_TYPE_ = 'CORR'` observations.

2. A DATA step, using the DATA Step Graphics Interface, determines how big to make the graphical plot.

3. PROC PLOT determines where to position the point labels. The results are sent to output SAS data sets using ODS. By default, if some of the point label characters are hidden, the `%PlotIt` macro recreates the printer plot with a larger line and page size, and hence creates more cells and more room for the labels.

4. The PROC PLOT output data sets are read, and information from them and the preprocessed data set are combined to create an Annotate data set. The label position information is read from the PROC PLOT output, and all of the symbol, size, font, and color information is extracted from the preprocessed data set. The Annotate data set contains all of the instructions for drawing the axes, ticks, tick marks, titles, point symbols, point labels, axis labels, and so on.

5. The Annotate data set is displayed with the GANNO procedure. The `%PlotIt` macro does not use PROC GPLOT.

With the `%PlotIt` macro, you can:

- display plots and create graphics stream files and `gout=` entries
- easily display the results of correspondence analysis, multidimensional preference analysis, preference mapping, multidimensional scaling, regression analysis, and density estimation
- use single or multi-character symbols and control their color, font, and size
- use multi-character point labels and control their color, font, and size
- use fixed, variable, and random colors, and use colors to display changes in a third dimension
- automatically determine a good line size, page size, and list of point label placements
- automatically equate the axes for all devices
- control the colors, sizes, fonts, and general appearance of all aspects of the plot

- pre- and post-process the data
- specify many `goptions`

Since `%PlotIt` is a macro, you can modify it, change the defaults, add new options, and so on. The `%PlotIt` macro is heavily commented to make it easier for you to modify it to suit your needs. There is substantial error checking and options to display intermediate results for debugging when you do not get the results you expect. Furthermore, you have complete access to all of the data sets it creates, including the preprocessed version of the input and the Annotate data set. You can modify the results by editing the Annotate and preprocessed data sets.

Changes and Enhancements

The main enhancement is the `%PlotIt` macro has been modified to produce plots that look more like graphs created by ODS Graphics. Primarily, this means that the appearance of the graph is at least in part sensitive to information in ODS styles. You can specify `style=` as an option on any ODS destination statement, for example: `ods html style=statistical;`. By default, the plots are produced using a default style, which depends on the destination. For the HTML destination, the default style is literally named `default`. If you specify `options nogstyle;`, then style information is not used, and the macro should do what it did previously. However, the previous output, particularly the fonts and the colors, are not as nice as the newer style-sensitive appearance.

Examples

This section contains examples of some of the capabilities of the `%PlotIt` macro. Rather than interpreting the plots or discussing the details of the statistical analyses, this section concentrates on showing what the `%PlotIt` macro can do. Most of the examples are based on SAS/STAT example data sets. Data for all of the examples can be found in the SAS/STAT sample program `plotitex.sas`.

Example 1: Principal Components of Mammal's Teeth
Principal component analysis computes a low-dimensional approximation to a set of data. Principal components are frequently displayed graphically. This example is based on the Mammal's Teeth data set. To perform a principal component analysis, specify:

```
proc princomp data=teeth out=scores(keep=prin1 prin2 mammal);
   title "Principal Components of Mammals' Teeth";
   run;

%plotit();
```

The plot is shown in Figure 1. No options were specified in the `%PlotIt` macro, so by default a plot is constructed from the first two numeric variables and the last character variable in the last data set created. The `%PlotIt` macro displayed the following information in the log:

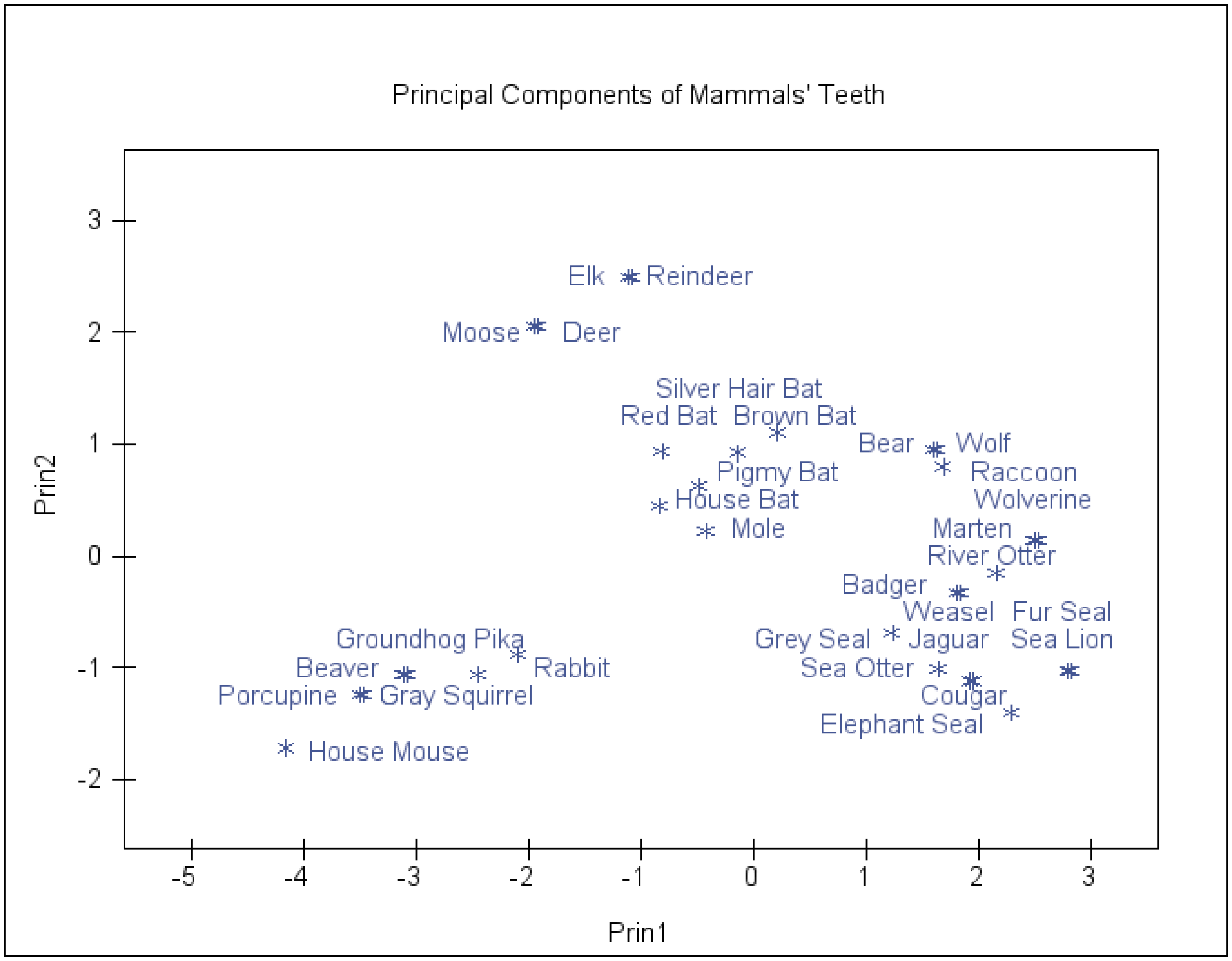


Figure 1. Principal Components

```
                 Iterative Scatter Plot of Labeled Points Macro

             Iteration    Place    Line Size    Page Size    Penalty
             -------------------------------------------------------
                 1          2         65           45           34
                 2          3         80           50            0
```

The following code will create the printer plot on which the graphical plot is based:

```
options nonumber ls=80 ps=50;
proc plot nolegend formchar='|----|+|---' data=preproc vtoh=2;
   plot Prin2 * Prin1 $ mammal = _symbol_ /
        haxis=by 1 vaxis=by 1 box list=1
        placement=((h=2 -2 : s=right left) (v=1 to 2 by alt * h=0 -1 to -10
        by alt));
   label Prin2 = '#' Prin1 = '#';
   run; quit;

The plot was created with the following goptions:

goptions reset=goptions erase hpos=99 vpos=34 hsize=11.71in vsize=7.98in
         device=WIN;

The OUT=anno Annotate data set has 148 observations.
The PLOTIT macro used 1.7seconds to create OUT=anno.
```

The iteration table shows that the `%PlotIt` macro tried twice to create the plot, with line sizes of 65 and 80. It stopped when all point label characters were plotted with zero penalty.[*] The `%PlotIt` macro displays PROC PLOT code for the printer plot, on which the graphical plot is based. It also displays the `goptions` statement that was used with PROC GANNO.[†]

There are several notable features of the plot in Figure 1.

1. Symbols for several pairs of points, such as Elk and Reindeer, are coincident. By default, the `%PlotIt` macro slightly offsets coincident symbols so that it is clear that more than one point maps to the same location.
2. Point labels map into discrete rows, just as they would in a printer plot produced by PROC PLOT. However, unlike printer plots, the `%PlotIt` macro uses proportional fonts.
3. Symbols are not restricted to fixed cells. Their mapping is essentially continuous, more like PROC GPLOT's than PROC PLOT's.
4. A fixed distance represents the same data range along both axes, which means the axes are equated so that distances and angles will have meaning. In contrast, procedures such as PLOT and GPLOT fill the available space by default, so the axes are not equated.

Example 2: Principal Components of Crime Rates
A typical plot has for each point a single-character symbol and a multi-character label; however, this is not required. This example is based on the Crime data set. The point labels are state names, and the symbol for each label is a two-character postal code.

[*] Penalties accrue when point labels are nonoptimally placed, such as when two label characters map to the same location. PROC PLOT tries to minimize the penalties for all point labels. See PROC PLOT documentation for more information.

[†] This code could be different depending on your device. By default, the macro uses your default device.

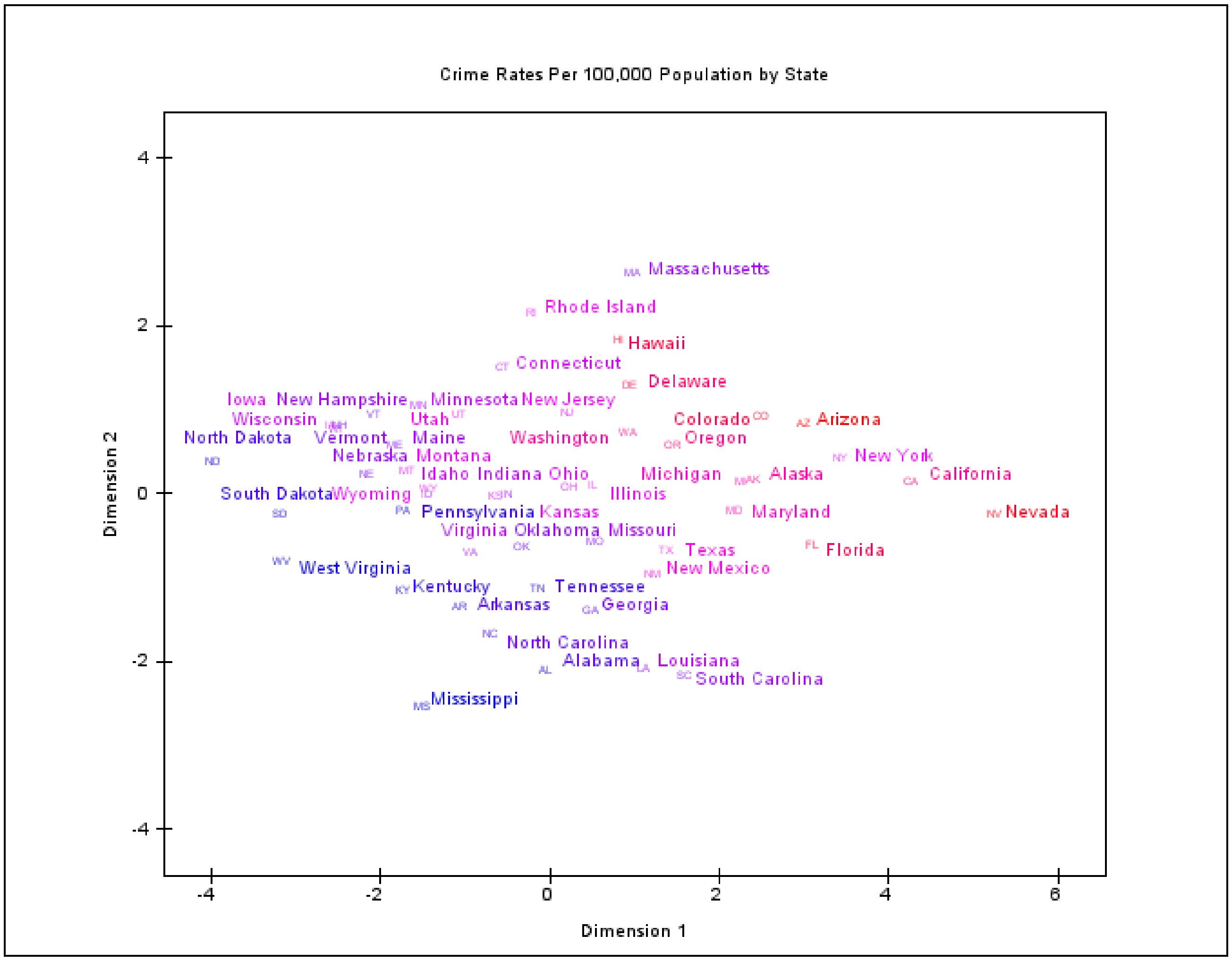


Figure 2. Principal Components with the Third Component "Painted"

```
proc princomp data=crime out=crime2;
   title 'Crime Rates Per 100,000 Population by State';
   run;

%plotit(data=crime2, plotvars=prin2 prin1,
        symvar=postcode, symlen=2, symsize=0.6, paint=larceny,
        labelvar=state, label=typical)
```

This plot request specifies:

- the input data set: `crime2`
- the y-axis and x-axis variables: `prin2` and `prin1`
- the symbol variable: `postcode`
- the number of symbol characters: 2
- the size of the symbol font in the plot: 0.6
- the colors are based on the variable: `larceny`
- the point label variable: `state`
- the typical method of generating variable labels for the plot axes

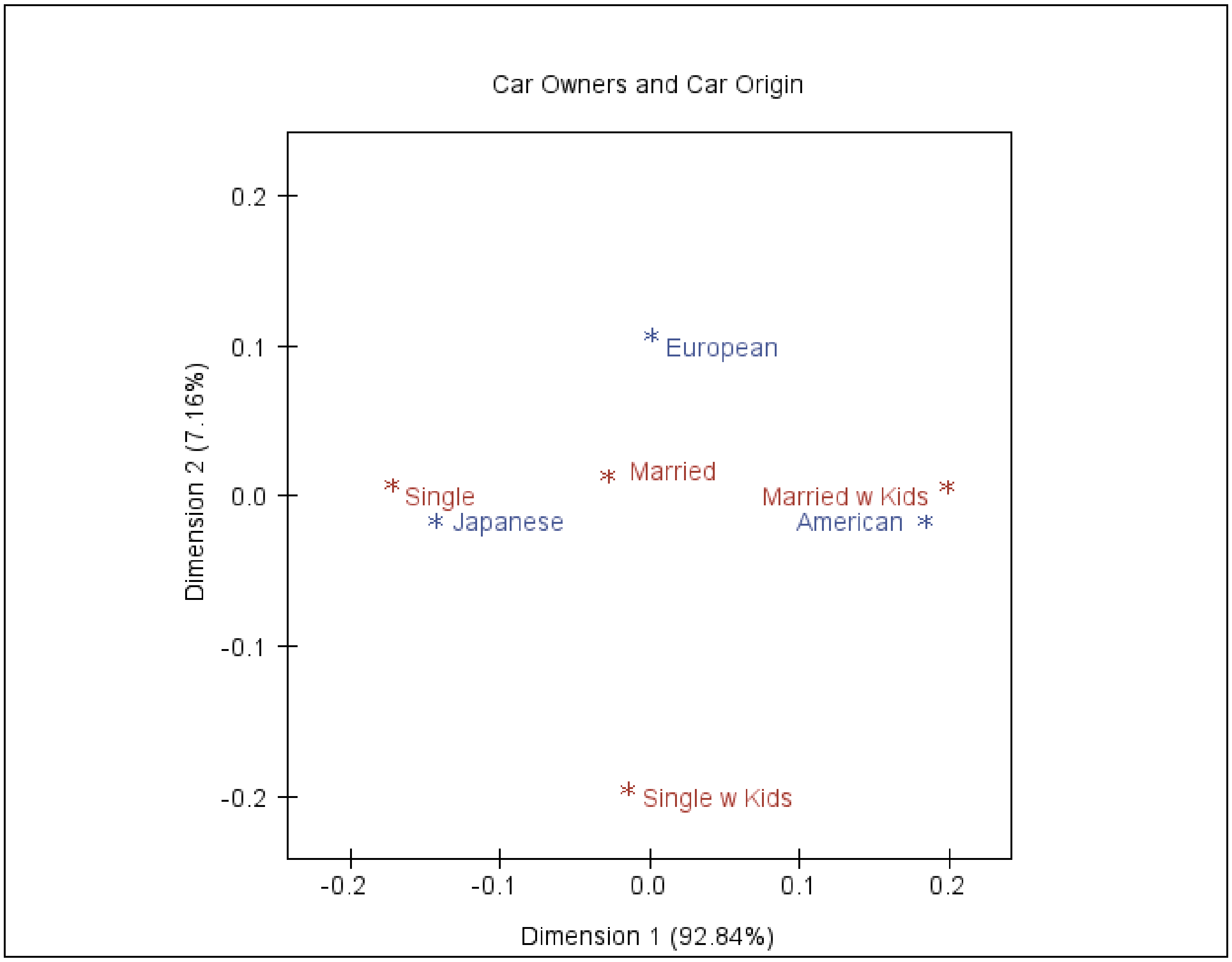


Figure 3. Simple Correspondence Analysis

A symbol size of 0.6 instead of the normal 1.0 is specified to make the symbol small, because two characters are mapping to a location where there is usually just one. The option `paint=larceny` creates interpolated label and symbol colors, by default between blue, magenta, and red, so that states that have a low larceny rate are blue and high-rate states are red. `Label=typical` for variables `prin2` and `prin1` generates the following label statement:

```
label prin2 = 'Dimension 2' prin1 = 'Dimension 1';
```

This plot request is much more complicated than most. Often, you need to specify only the type of analysis that generated the data set. The plot is shown in Figure 2.

Examples 3 & 4: Correspondence Analysis of the Car Ownership Survey
Correspondence analysis graphically displays crosstabulations. These examples use the Car Survey data. To perform a correspondence analysis, specify:

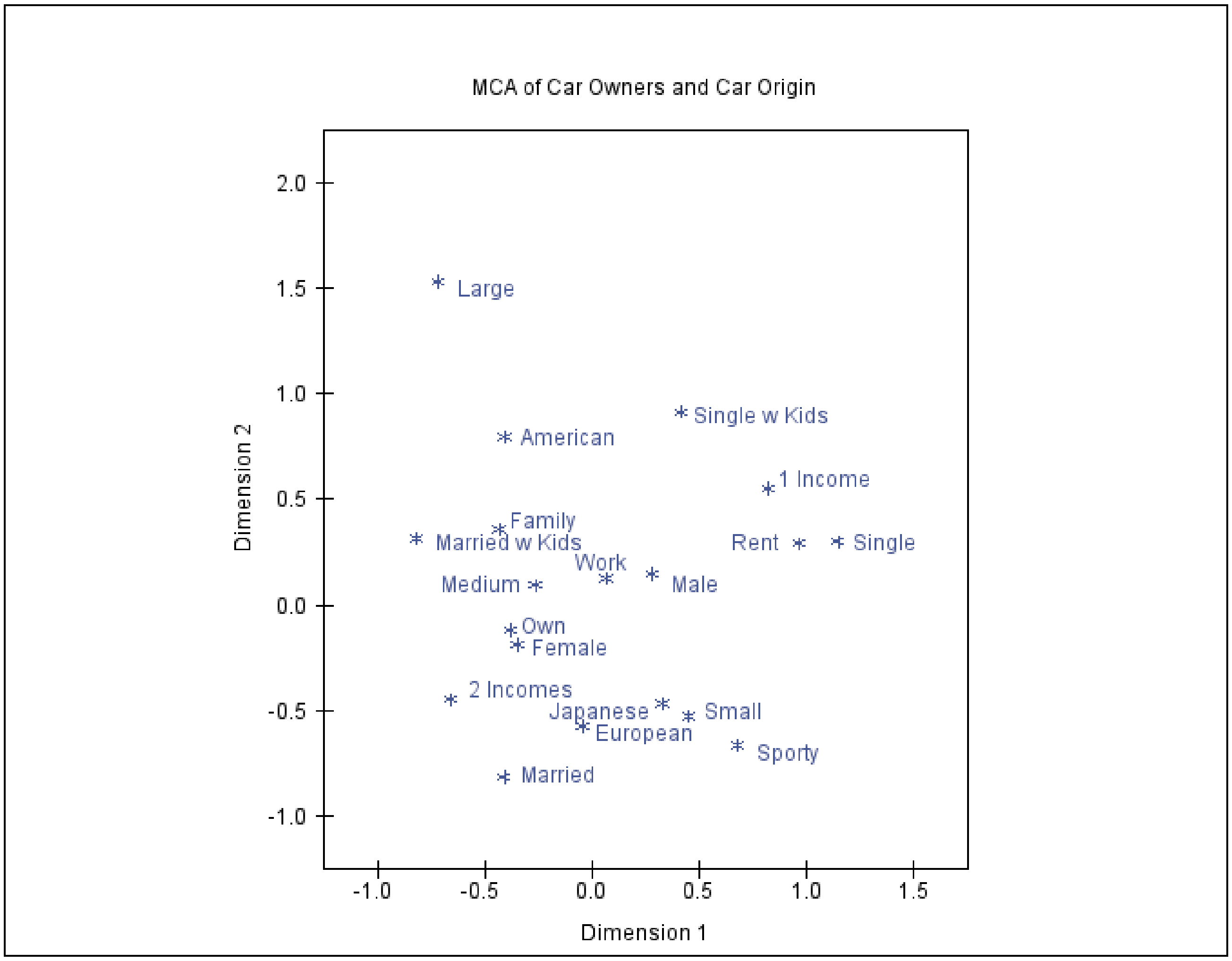


Figure 4. Multiple Correspondence Analysis

```
proc corresp data=cars outc=coors;
   title 'Car Owners and Car Origin';
   tables marital, origin;
   run;

%plotit(data=coors,datatype=corresp)
```

The plot is shown in Figure 3. With `datatype=corresp`, the `%PlotIt` macro automatically incorporates the proportion of inertia* into the axis labels and plots the row points in red and the column points in blue.

For a multiple correspondence analysis, specify:

```
proc corresp mca observed data=cars outc=coors;
   title 'MCA of Car Owners and Car Origin';
   tables origin size type income home marital sex;
   run;
```

*Inertia in correspondence analysis is analogous to variance in principal component analysis.

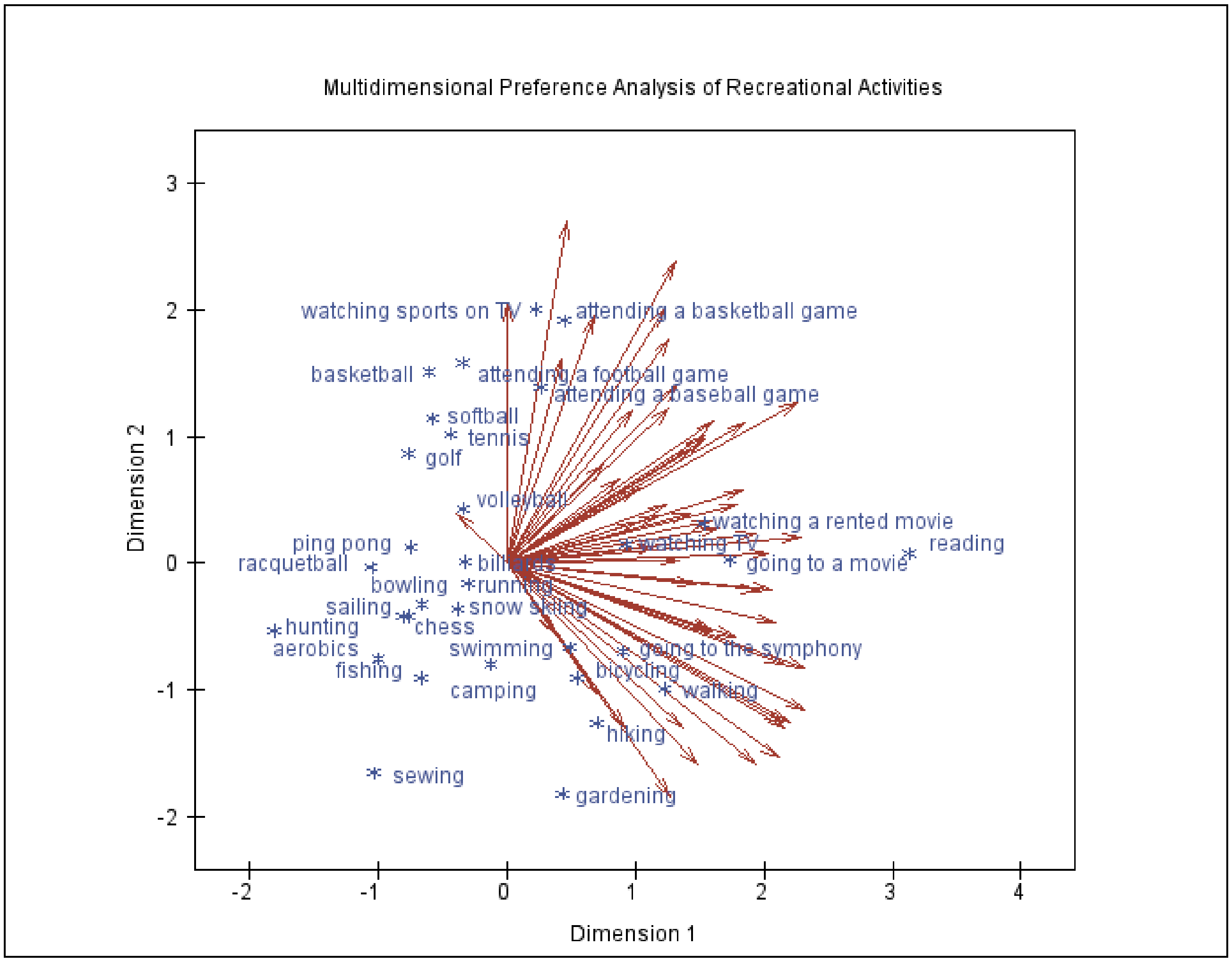


Figure 5. Multidimensional Preference Analysis

```
%plotit(data=coors,datatype=mca)
```

The plot is shown in Figure 4.

Alternatively, you can specify the `source` option to create a variable `_var_`, which you can use as a type variable to get different colors for each set of categories corresponding to each of the input variables, for example as follows:

```
proc corresp mca observed data=cars outc=coors;
   title 'MCA of Car Owners and Car Origin';
   tables origin size type income home marital sex;
   run;

%plotit(data=coors, plotvars=dim2 dim1, typevar=_var_)
```

The results of this step are not shown.

Examples 5 & 6: Multidimensional Preference Analysis of Recreational Activities
Multidimensional preference analysis is a variant on principal component analysis that simultaneously

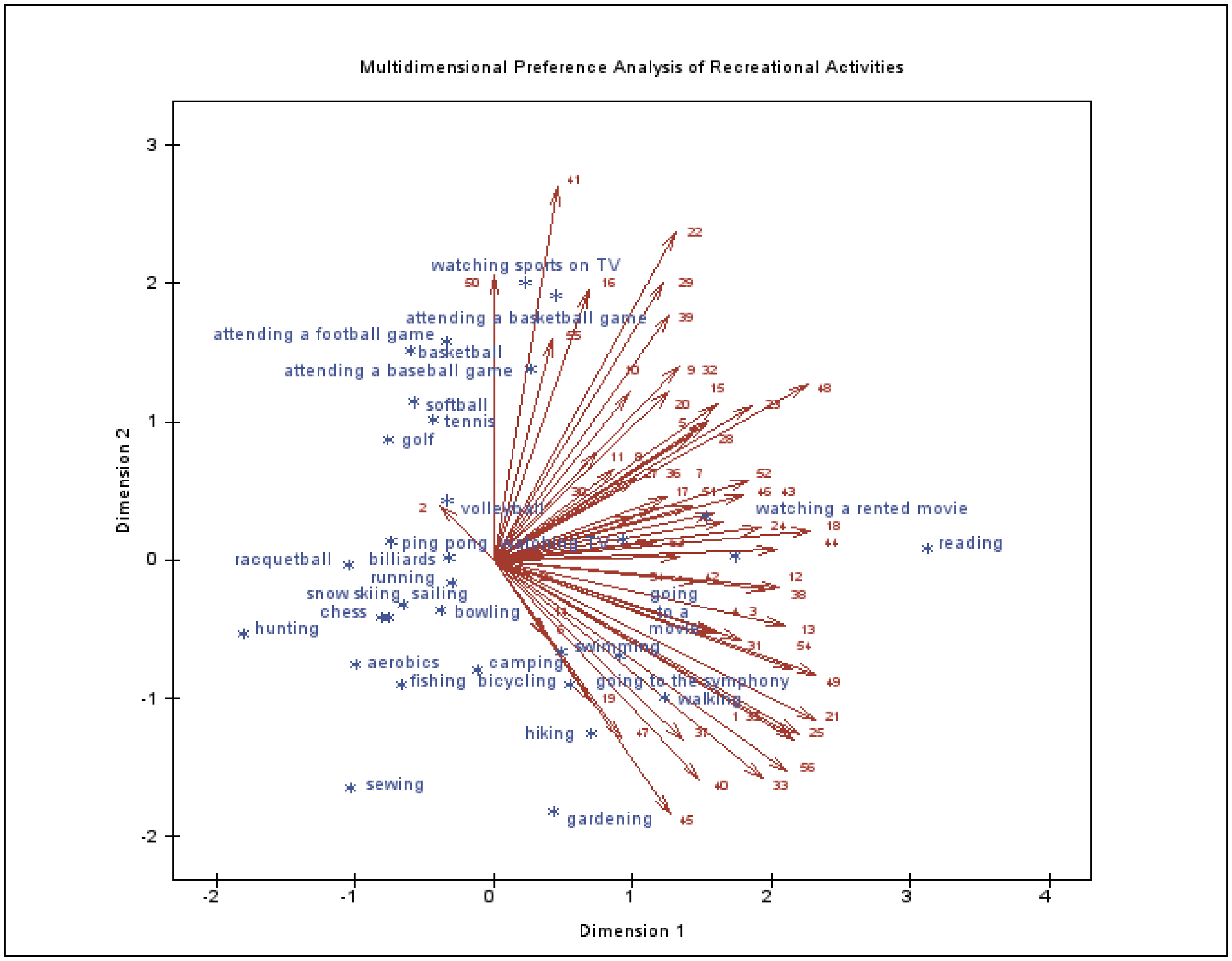


Figure 6. Multidimensional Preference Analysis

displays people and their preferences for objects. Each person is a variable in the input data set, and each object is a row. Each person is represented in the plot as a vector that points in approximately the direction of his or her most preferred objects. These examples use the Preferences for Recreational Activities data set. For multidimensional preference analysis, specify:

```
proc prinqual data=recreate out=rec mdpref rep;
   title1 'Multidimensional Preference Analysis of Recreational Activities';
   transform identity(sub1-sub56);
   id activity;
   run;

%plotit(data=rec,datatype=mdpref 3)
```

The plot is shown in Figure 5. With `datatype=mdpref`, the `%PlotIt` macro automatically displays the people as vectors and the activities as points (based on the `_TYPE_` variable). The `3` after the MDPREF is a scaling factor for the vectors. The lengths of all the vectors are increased by a factor of three to create a better graphical display. You can also label the vectors by specifying:

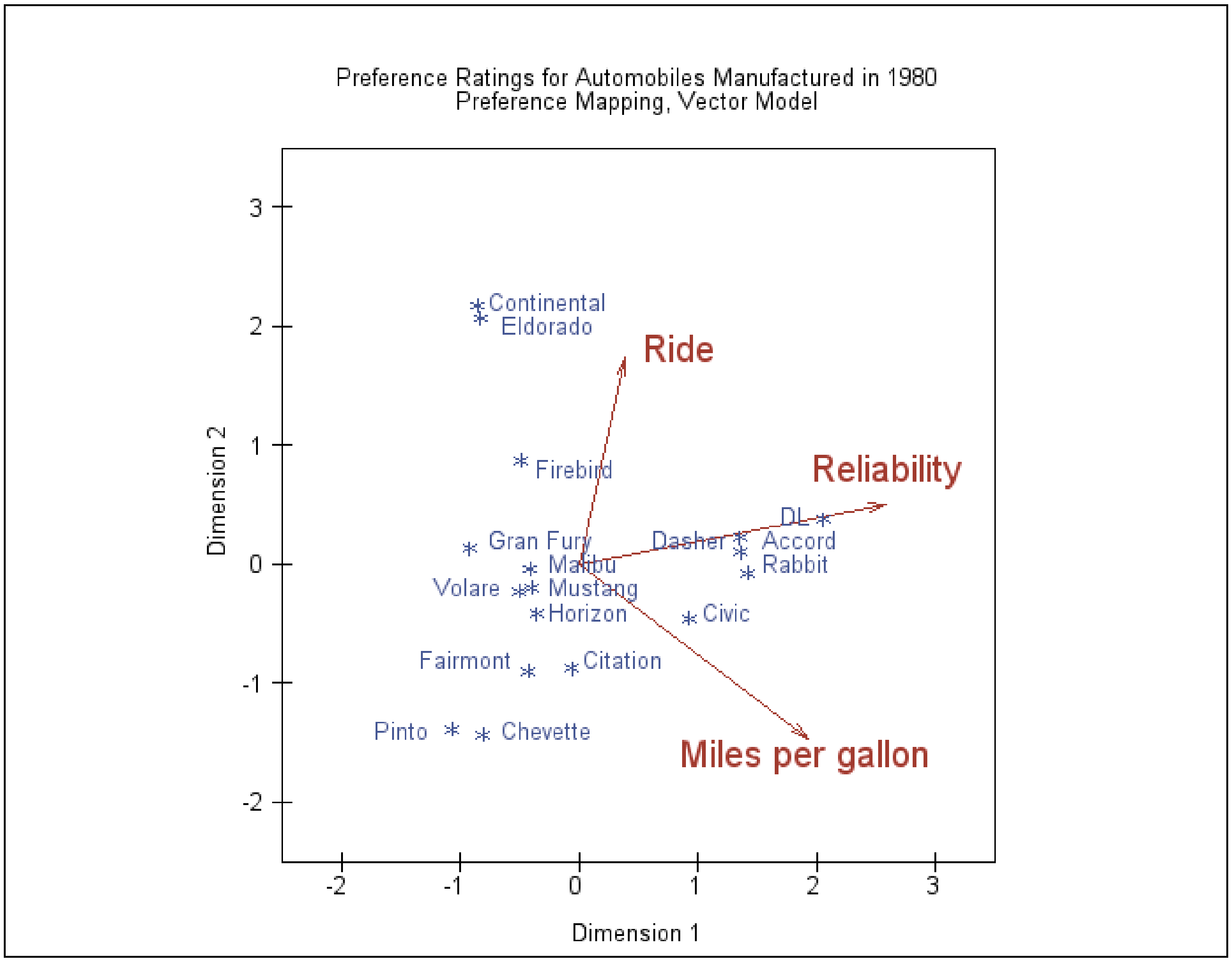


Figure 7. Preference Mapping, Vector Model

```
    %plotit(data=rec,datatype=mdpref2 3)
```

MDPREF2 specifies a MDPREF analysis with labeled vectors (the `2` means labels *too*). This plot is not shown because in this input data set, each subject is identified by a variable name of the form `sub1`, `sub2`, ..., and the graphical display looks cluttered with all those `sub`'s. The default point label variable is the ID statement variable `activity`, because it is the last character variable in the data set. PROC PRINQUAL fills in this variable for the `_TYPE_ = 'CORR'` observations (the people that plot as vectors) with the variable names: `sub1-sub56`. You can preprocess the input data set directly in the `%PlotIt` macro to remove the `sub`'s as follows:

```
    %plotit(data=rec,datatype=mdpref2 3,
            adjust1=%str(if _type_ = 'CORR' then
                        activity = substr(activity,4);))
```

The plot is shown in Figure 6. The `adjust1` option adds DATA step statements to the end of the preprocessing step. By default, the `%PlotIt` macro tries to position the vector labels outward, not between the vector head and the origin.

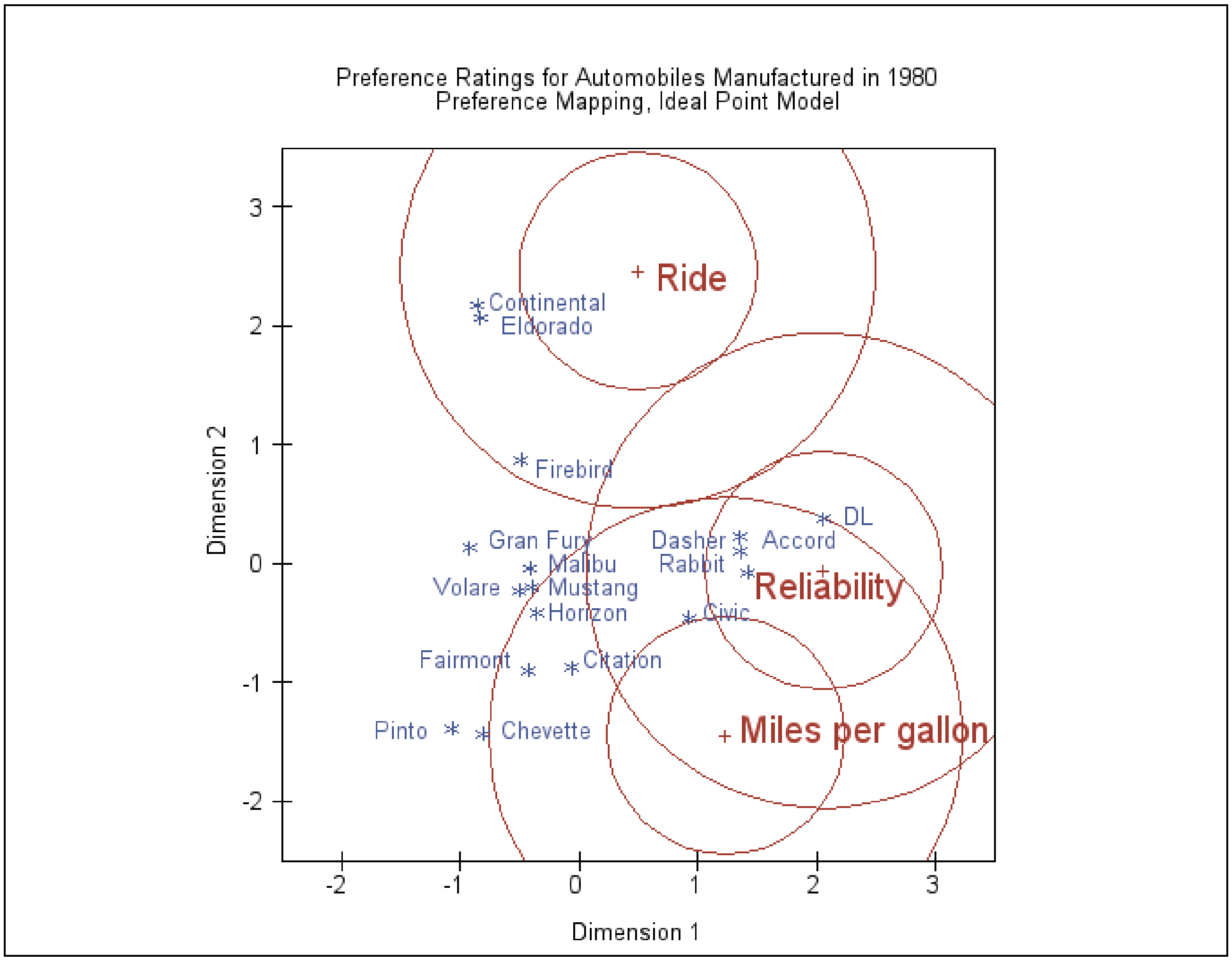


Figure 8. *Preference Mapping, Ideal Point Model*

Examples 7 & 8: Preference Mapping of Cars
Preference mapping simultaneously displays objects and attributes of those objects. These examples use the Car Preference data set to illustrate preference mapping. The following code fits a preference mapping vector model:

```
* Compute Coordinates for a 2-Dimensional Scatter plot of Cars;
proc prinqual data=carpref out=presults(drop=judge1-judge25) n=2
              replace mdpref;
   title 'Preference Ratings for Automobiles Manufactured in 1980';
   id model mpg reliable ride;
   transform identity(judge1-judge25);
   run;
```

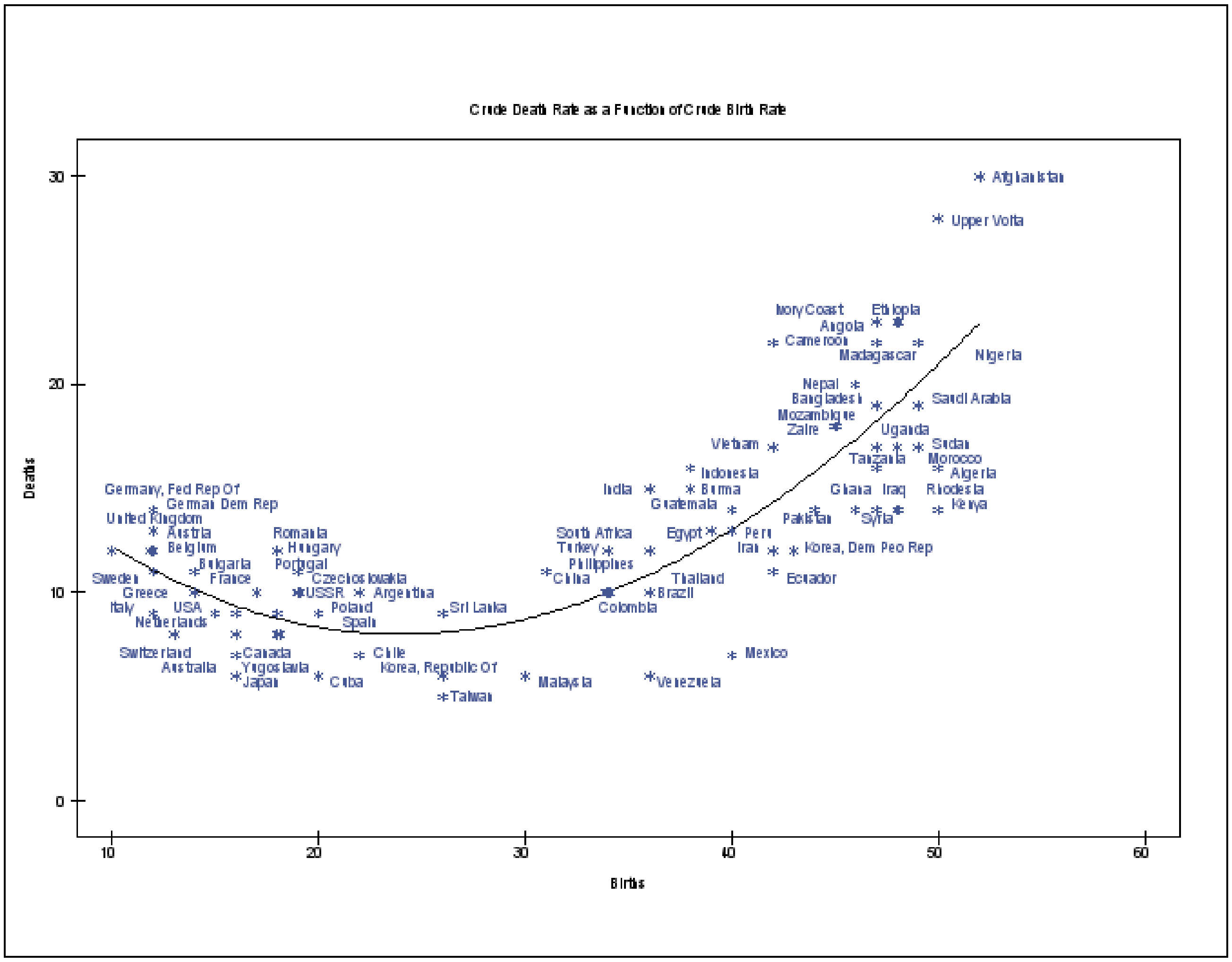


Figure 9. Spline Regression Model

```
* Compute Endpoints for Vectors;
proc transreg data=presults;
   title2 'Preference Mapping, Vector Model';
   model identity(mpg reliable ride)=identity(prin1 prin2);
   output tstandard=center coefficients replace out=vector;
   id model;
   run;

%plotit(data=vector,datatype=vector 2.5)
```

The plot is shown in Figure 7. Each attribute is represented as a vector that points in approximately the direction of the objects with larger values of the attribute. The `datatype=vector 2.5` option requests the vector model, with the vectors stretched by a factor of 2.5. Alternatively, you can represent attributes as points by specifying:

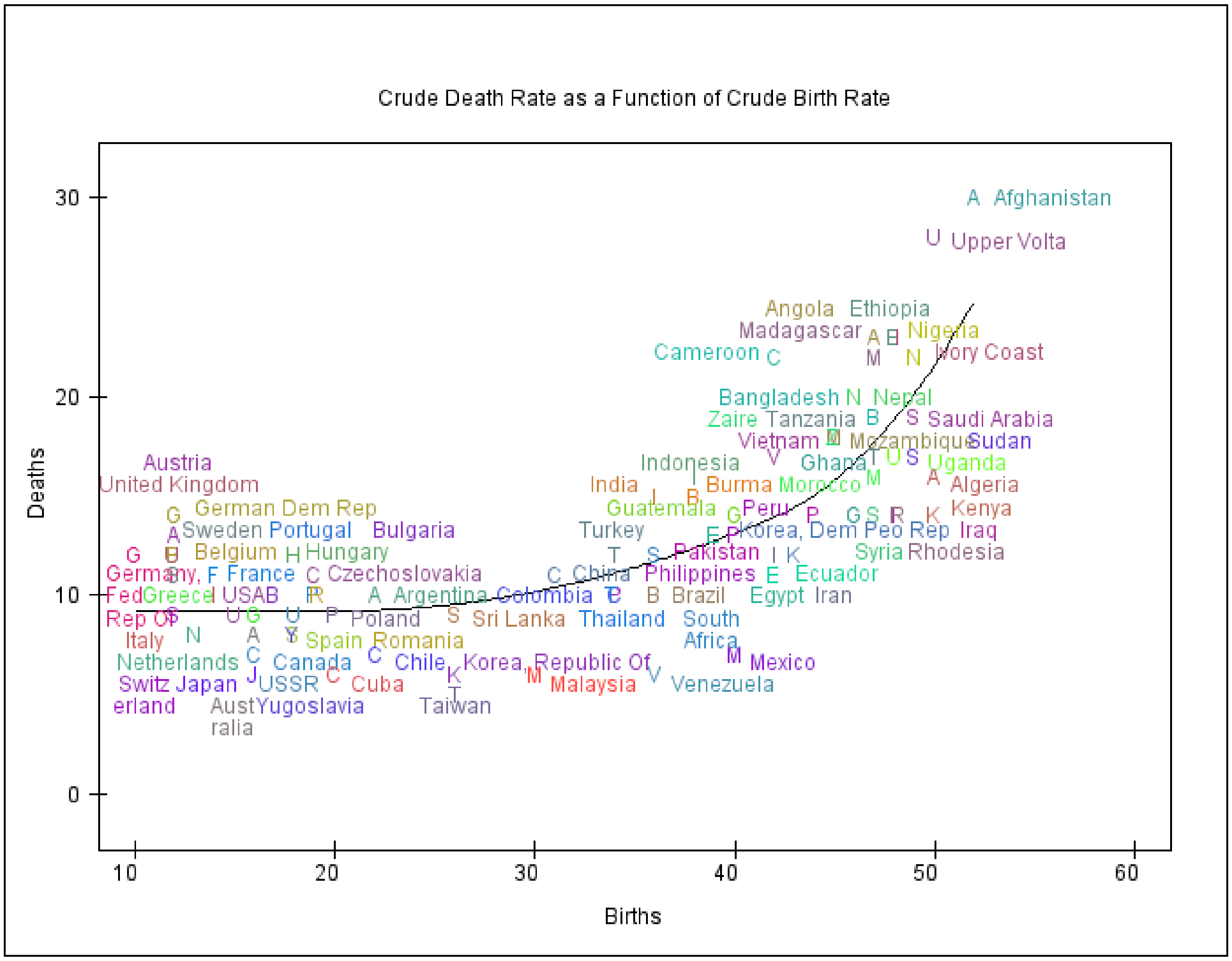


Figure 10. Spline Regression Model

```
* Compute Ideal Point Coordinates;
proc transreg data=presults;
   title2 'Preference Mapping, Ideal Point Model';
   model identity(mpg reliable ride)=point(prin1 prin2);
   output tstandard=center coordinates replace out=ideal;
   id model;
   run;

%plotit(data=ideal,datatype=ideal,antiidea=1)
```

The plot is shown in Figure 8. The option `datatype=ideal` requests a preference mapping, with each attribute represented as an ideal point. Circles are drawn to show distances between the cars and the ideal points, which are locations of hypothetical cars that have the ideal amount of the attribute. The `antiidea=1` option specifies how anti-ideal points are recognized.* By default, the labels for the attributes are larger than the other point labels and hence sometimes extend slightly beyond the plot. This happened with "Miles per gallon" in Figure 8. You can move the label up one character unit and

*Anti-ideal points have their signs wrong, so the macro must reverse them before plotting. When small ratings are good, specify `antiidea=-1`, and when small ratings are not good, specify `antiidea=1`.

to the left 12 character units by adding the following option:

```
adjust4=%str(if text =: 'Miles' then do; y = y + 1; x = x - 12; end;)
```

The `adjust4` option adds DATA step statements to the end of the final Annotate DATA step. The `%PlotIt` macro has no sense of aesthetics; sometimes a little human intervention is needed for the final production plots.

Examples 9 & 10: Curve Fitting of Birth and Death Rates
It is often useful to display a set of points along with a regression line or nonlinear function. The `%PlotIt` macro can fit and display lines and curves (and optionally display the regression and ANOVA table). These examples use the Vital Statistics data set. The following requests a cubic-polynomial regression function:

```
title 'Crude Death Rate as a Function of Crude Birth Rate';

%plotit(data=vital,vtoh=1.75,datatype=curve2)
```

The plot is shown in Figure 9. The option `vtoh=1.75` specifies the PROC PLOT aspect ratio (vertical to horizontal). The default is 2.0. Smaller values create plots with more cells for label characters, which is helpful when the point cloud is relatively dense. The option `datatype=curve2` instructs the `%PlotIt` macro to fit a curve and have the point labels avoid the curve (the `2` means label avoidance *too*).

You can control the type of curve. The `%PlotIt` macro uses PROC TRANSREG to fit the curve, and you can specify PROC TRANSREG options. For example, to request a monotone spline regression function with two knots, specify:

```
%plotit(data=vital,datatype=curve,bright=128,maxiter=4,
        symvar=country,regfun=mspline,nknots=2)
```

The plot is shown in Figure 10. There are several differences between Figures 9 and 10, in addition to the difference in the regression function. The option `datatype=curve` was specified, not `datatype=curve2`, so there is more overlap between the point labels and the curve. For each point in the plot, the plotting symbol is the first letter of the country and the point label is the country. Each label/symbol pair is a different random color with brightness (average RGB or red-green-blue value) of 128. These options make it much easier to find the symbol that corresponds to each label. Also, the default `vtoh=2` was used to decrease the number of cells and make the labels larger. With these data, the penalty sum at iteration four is eight. Specifying `maxiter=4` prevents the algorithm from reaching iteration 5, which prevents the line size from increasing from 125 to 150. This also makes the labels larger. The price is that some label characters collide (for example, “Germany” and “S”) and the plot looks more cluttered because there are fewer cells with white space.

Availability

If your site has installed the latest autocall libraries supplied by SAS and uses the standard configuration of SAS supplied software, you need only to ensure that the SAS system option `mautosource` is in effect to begin using autocall macros. That is, the macros do *not* have to be included (for example with a `%include` statement). They can be called directly. For more information about autocall libraries, see

SAS Macro Language: Reference and your host documentation. Also see pages 803–805, in the macros chapter for more information.

If you do not have the latest autocall macros installed, you can get them from the Web `http://support.sas.com/resources/papers/tnote/tnote_marketresearch.html`.

Base SAS and SAS/GRAPH software are required to run the `%PlotIt` macro. The `datatype=curve` and `datatype=curve2` options use PROC TRANSREG, which is in SAS/STAT. All of the other `datatype=` values assume an input data set in a form created by a SAS/STAT procedure.

Conclusions

The `%PlotIt` macro provides a convenient way to display the results from many types of data analyses. Usually, only a small number of options are needed; the macro does the rest. The `%PlotIt` macro does not replace procedures like GPLOT and PLOT. Instead, it makes it easy to generate many types of plots that are extremely difficult to produce with standard procedures.

Appendix: ODS Graphics

With SAS Version 9.2, ODS Graphics automatically does much of what the `%PlotIt` macro was originally designed to do, and usually, ODS Graphics does it better and more conveniently. This appendix shows how to generate some of the same plots as were shown in the main body of this paper, and a few additional plots as well. All plots in this appendix are generated with ODS Graphics. Most of these plots come out of a procedure by default. Some must be requested with a `plots=` option (e.g. `plots=score` in PROC PRINCOMP). Others are directly produced by the SG (Statistical Graphics) plotting procedures SGRENDER and SGPLOT. You can use PROC SGRENDER and PROC SGPLOT on both raw data sets or output data sets produced by other procedures. This chapter also shows the ODS statements and options that are used for capturing plots in files so that they may later be included into a document (such as this one). Usually, of course, books show you the code to make a plot, not the full code needed to display this plot in this place in this document. This technique can be directly applied to other documents composed using LaTeX, and less directly in other documents.

The first statement that you run is as follows:

```
ods listing style=statistical gpath='png';
```

This opens the LISTING destination, which is open by default, specifies that output will be generated with the Statistical style, and specifies that the graphical output will go to a directory called 'png', which must be created as a subdirectory under the current working directory. The style controls the color scheme and the look and feel of the output. There are many styles, and the Statistical style is the one that is used with SAS/STAT documentation. The following statements produce Figure 11:

```
ods graphics / reset=index imagename="obsodspc1";
proc princomp data=teeth plots=score n=2;
   title "Principal Components of Mammals' Teeth";
   ods select scoreplot;
   id mammal;
   run;
```

The ODS Graphics plot in Figure 11 corresponds to the `%PlotIt` macro plot in Figure 1. The ODS Graphics statement specifies that plots are to be generated in the `png` directory, with a name constructed from the `imagename=` value, possibly followed by a numeric suffix if the step produces more than one plot, and the suffix '.png'. The generated graph name, including the path is `png/obsodspc1.png`. The file is included into the LaTeX document with the command `\includegraphics{png/obsodspc1.png}`. Before the first use of `includegraphics`, there is a single line to enable the `includegraphics` command: `\usepackage{graphicx}`. You can bring in the graph slightly more indirectly by defining a command `getgraph` in LaTeX as follows:

```
\newcommand{\getgraph}[1]{%
\begin{center}%
\includegraphics{/u/saswfk/text/book2006/png/#1.png}%
\end{center}}
```

Figure 11 was included into the document as follows:

```
\begin{figure}[t]
\getgraph{obsodspc1}
```

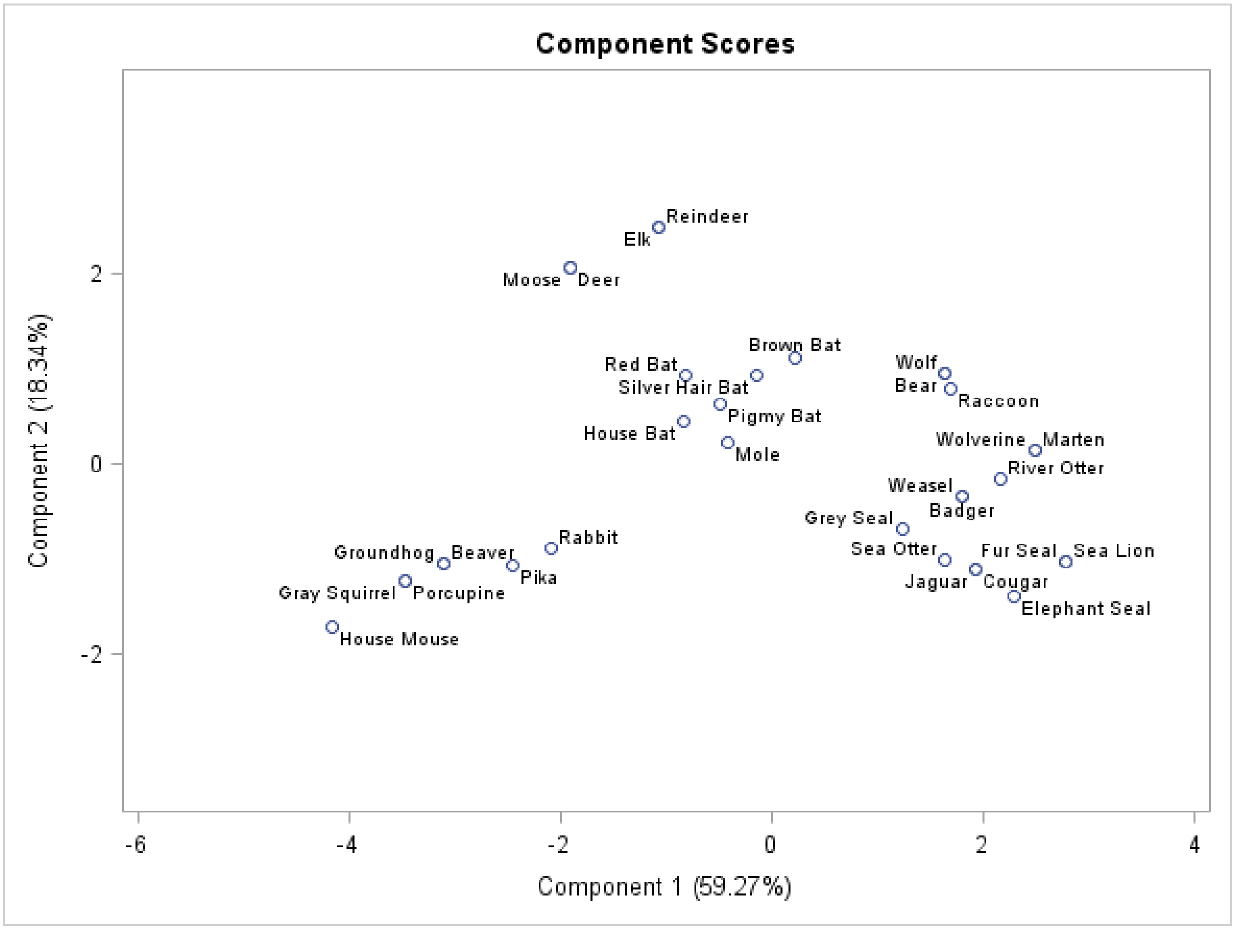


Figure 11.   Score Plot from PROC PRINCOMP

```
{\it Figure 11. \hspace{0.001in} Score Plot from PROC PRINCOMP}
\end{figure}
```

Let's return to the SAS statements again, which are as follows:

```
ods graphics / reset=index imagename="obsodspc1";
proc princomp data=teeth plots=score n=2;
   title "Principal Components of Mammals' Teeth";
   id mammal;
   run;
```

PROC PRINCOMP is run to perform the principal component analysis. The nondefault scores plot is requested by the `plots=score` option. The ODS SELECT statement specifies the plot that we want to see. This statement is not necessary, but it does exclude all other output and ensures that an integer need not be appended to the graph name. If the ODS SELECT statement had not been specified, and if the score plot were the second plot, then the plot names would be `obsodspc1.png` and `obsodspc11.png`, then the plot we need is `obsodspc11.png`. If the PROC PRINCOMP step is run again, without the `ods graphics / reset=index` statement, subsequent plots would be named `obsodspc1.png12`, `obsodspc13.png`, and so on. With the `ods graphics / reset=index` statement, the names are the same each time since the index is reset to zero or blank.

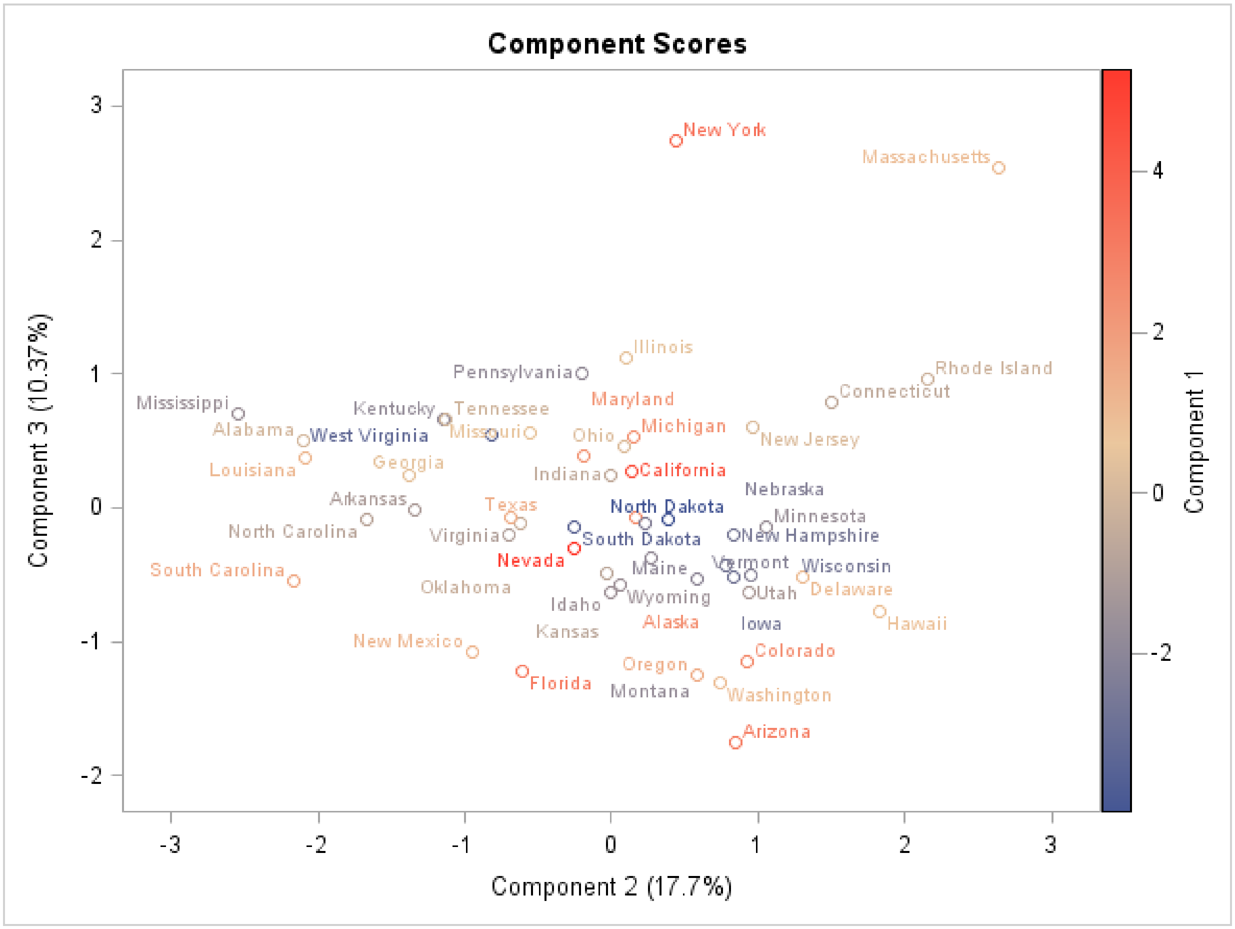


Figure 12. Painted Score Plot from PROC PRINCOMP

The following statements create Figure 12, which is in some ways similar to Figure 2:

```
ods graphics / reset=index imagename="obsodspc2";
proc princomp data=crime out=crime2 plots=score n=3;
   title 'Crime Rates Per 100,000 Population by State';
   ods select paintedscoreplot;
   id state;
   run;
```

Figure 12 consists of component 3 by component 2, colored by (or "painted" by) the values of component 1. In contrast, Figure 2 consists of component 2 by component 1, painted by the values of component 3. We can use ODS Graphics to make a plot like the one shown in Figure 2 with PROC TEMPLATE, the GTL (graphical template language), and PROC SGRENDER. This is a fairly straight-forward graphical template. It consists of a `define statgraph` and `end` block, which provides the name of the template. The template name, in this case `plot`, must be specified with PROC SGRENDER. There is a nested `begingraph/endgraph` block containing an `entrytitle` (plot title). Then there is a `layout overlay/endlayout` block. This is the outer shell most frequently used in making plots with SGRENDER, although other `layout` statements are possible. Inside the layout is a `scatterplot` statement that names the y axis variable, the x axis variable, the label variable, and the third variable whose values are shown with a gradient of colors. The `continuouslegend` statement produces the color "thermometer" legend for the statement named `'scores'`. The PROC SGRENDER step names the input data set and the template. The optional LABEL statement provides more descriptive labels for the axis variables. The following statements create Figure 13, which is similar to Figure 2:

```
proc template;
   define statgraph plot;
      begingraph;
         entrytitle 'Crime Rates Per 100,000 Population by State';
         layout overlay;
            scatterplot y=prin2 x=prin1 / datalabel=state
                        markercolorgradient=prin3 name='scores';
            continuouslegend 'scores' / title='Component 3';
            endlayout;
         endgraph;
      end;
   run;

ods graphics / reset=index imagename="obsodspc3";
proc sgrender data=crime2 template=plot;
   label prin1 = 'Component 1' prin2 = 'Component 2';
   run;
```

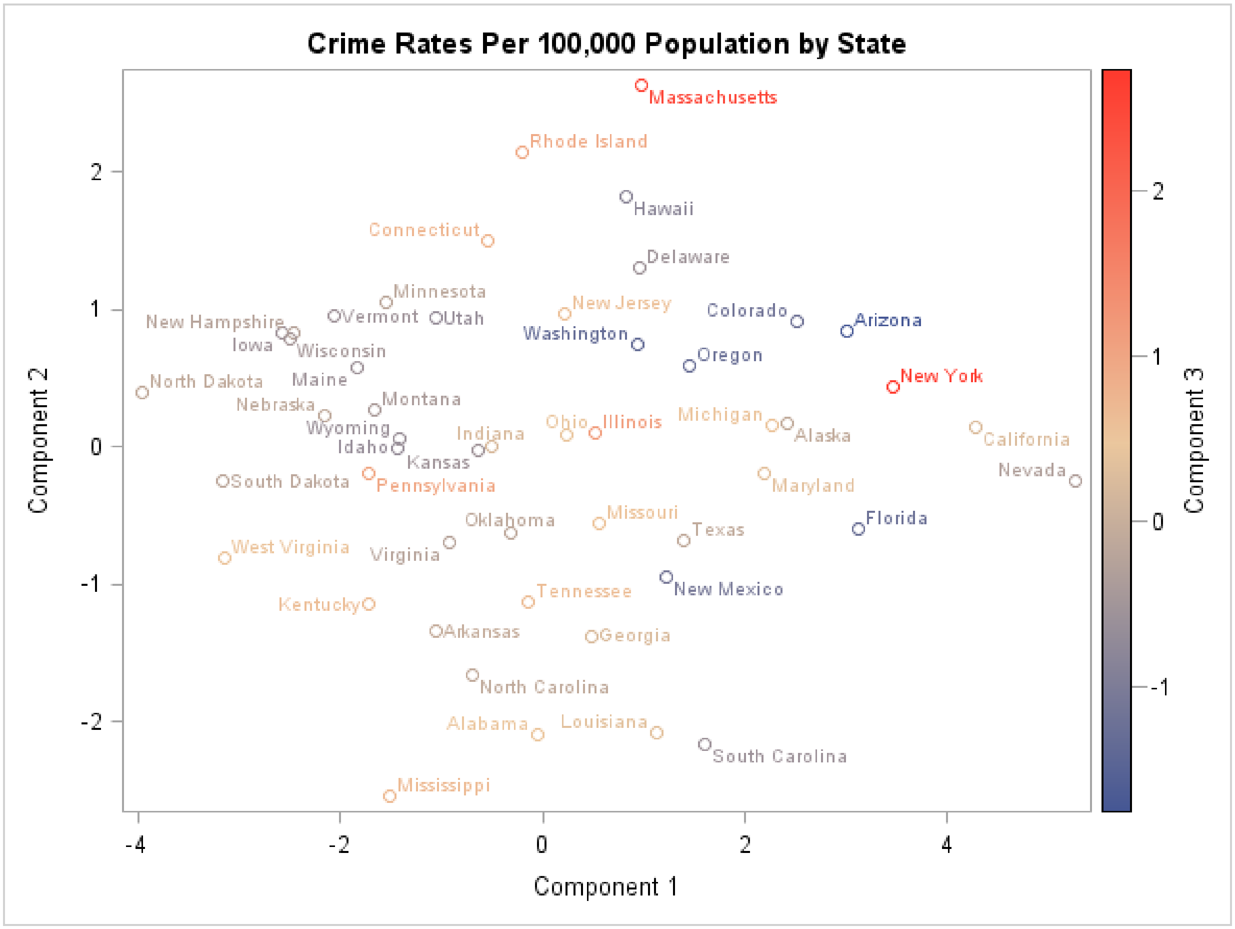


Figure 13. Painted Score Plot from PROCs TEMPLATE and SGRENDER

The following statements create Figure 14, which is similar to Figure 3:

```
ods graphics / reset=index imagename="obsodsca";
proc corresp data=cars outc=coors;
   title 'Car Owners and Car Origin';
   ods select configplot;
   tables marital, origin;
   run;
```

When ODS Graphics is enabled, PROC CORRESP automatically produces the correspondence analysis configuration plot.

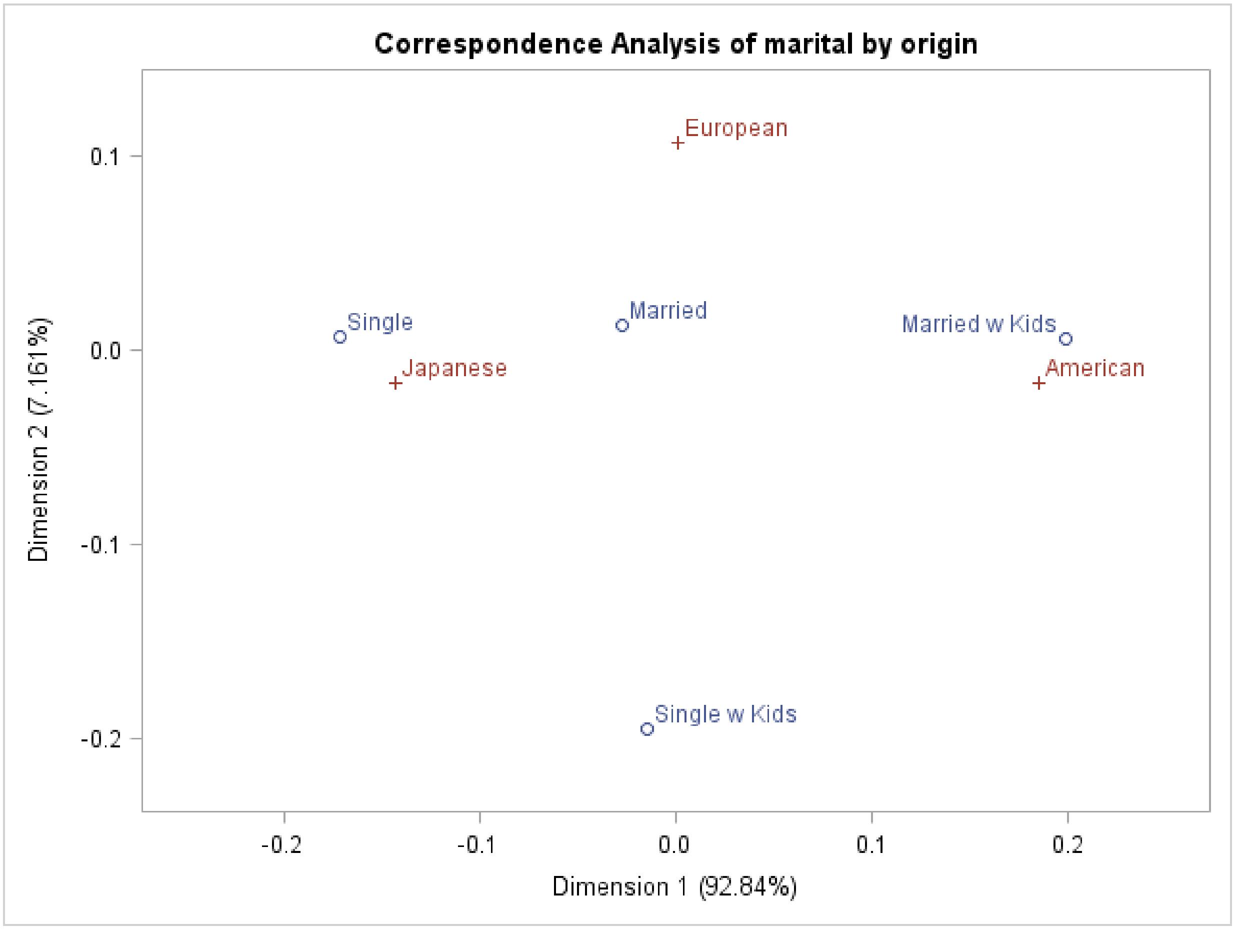


Figure 14. Simple Correspondence Analysis from PROC CORRESP

The following statements create Figure 15, which is similar to Figure 4:

```
ods graphics / reset=index imagename="obsodsmca";
proc corresp mca observed data=cars outc=coors;
   title 'MCA of Car Owners and Car Origin';
   ods select configplot;
   tables origin size type income home marital sex;
   run;
```

When ODS Graphics is enabled, PROC CORRESP automatically produces the multiple correspondence analysis configuration plot.

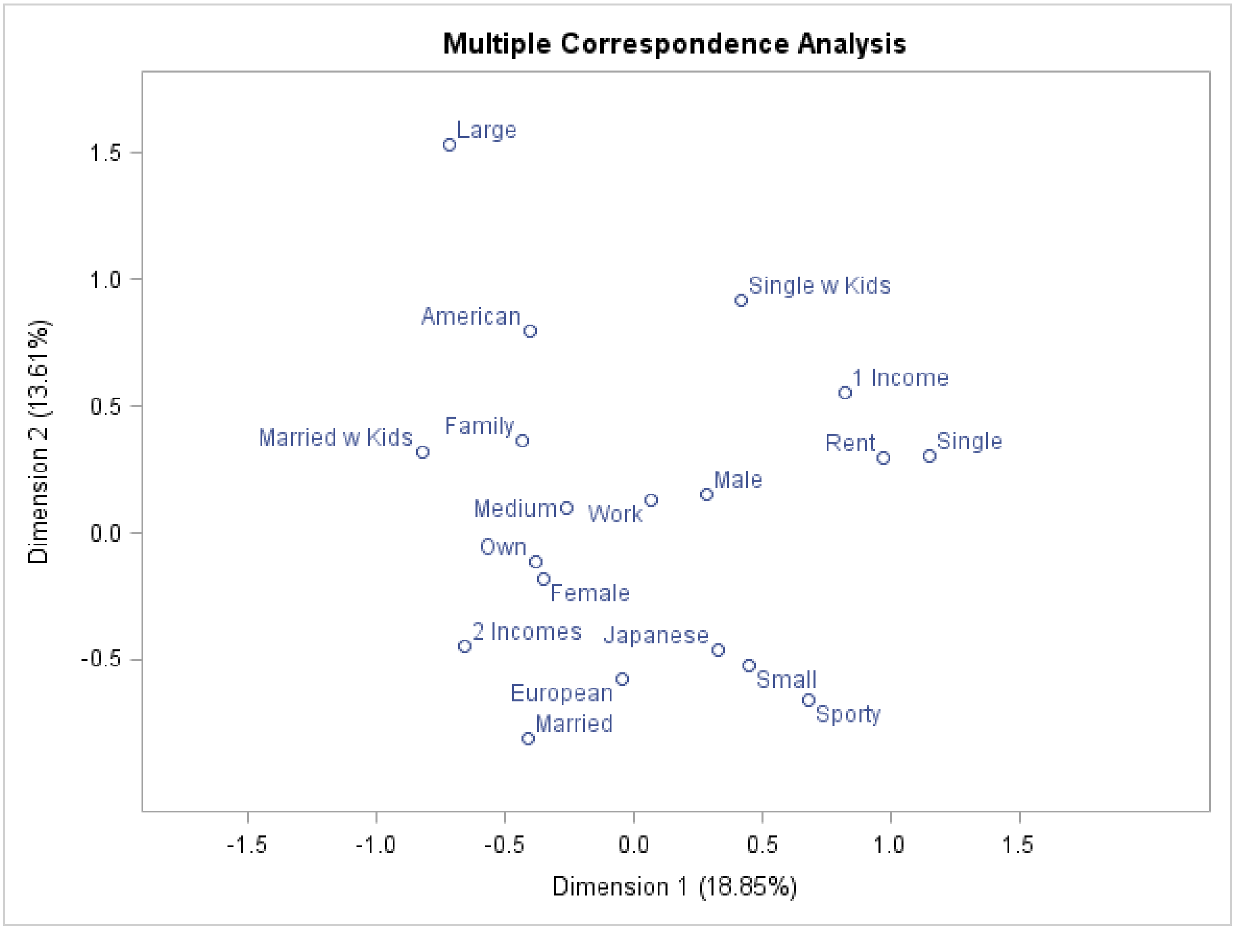


Figure 15. Multiple Correspondence Analysis from PROC CORRESP

The following statements create Figure 16, which is similar to Figure 6:

```
%macro ren(p,n); rename=(%do i = 1 %to &n; &p&i = "&i"n %end;) %mend;

options validvarname=any;
data rec2(%ren(sub,56)); set recreate; run;

ods graphics / reset=index imagename="obsodsmdp";
proc prinqual data=rec2 mdpref;
   title1 'Multidimensional Preference Analysis of Recreational Activities';
   ods select mdprefplot;
   transform identity('1'n-'56'n);
   id activity;
   run;
```

When ODS Graphics is enabled, PROC PRINQUAL and the `mdpref` option automatically produce the MDPREF plot. The variable names are displayed in the plot as labels for the vectors. In an MDPREF analysis, each column in the data set actually corresponds to a subject not a variable. This example uses a macro to rename the input variables from `sub1`, `sub2`, and so on to `1`, `2`, and so on to make a better graphical display—one that is not cluttered up by the prefix `sub` appearing 56 times.

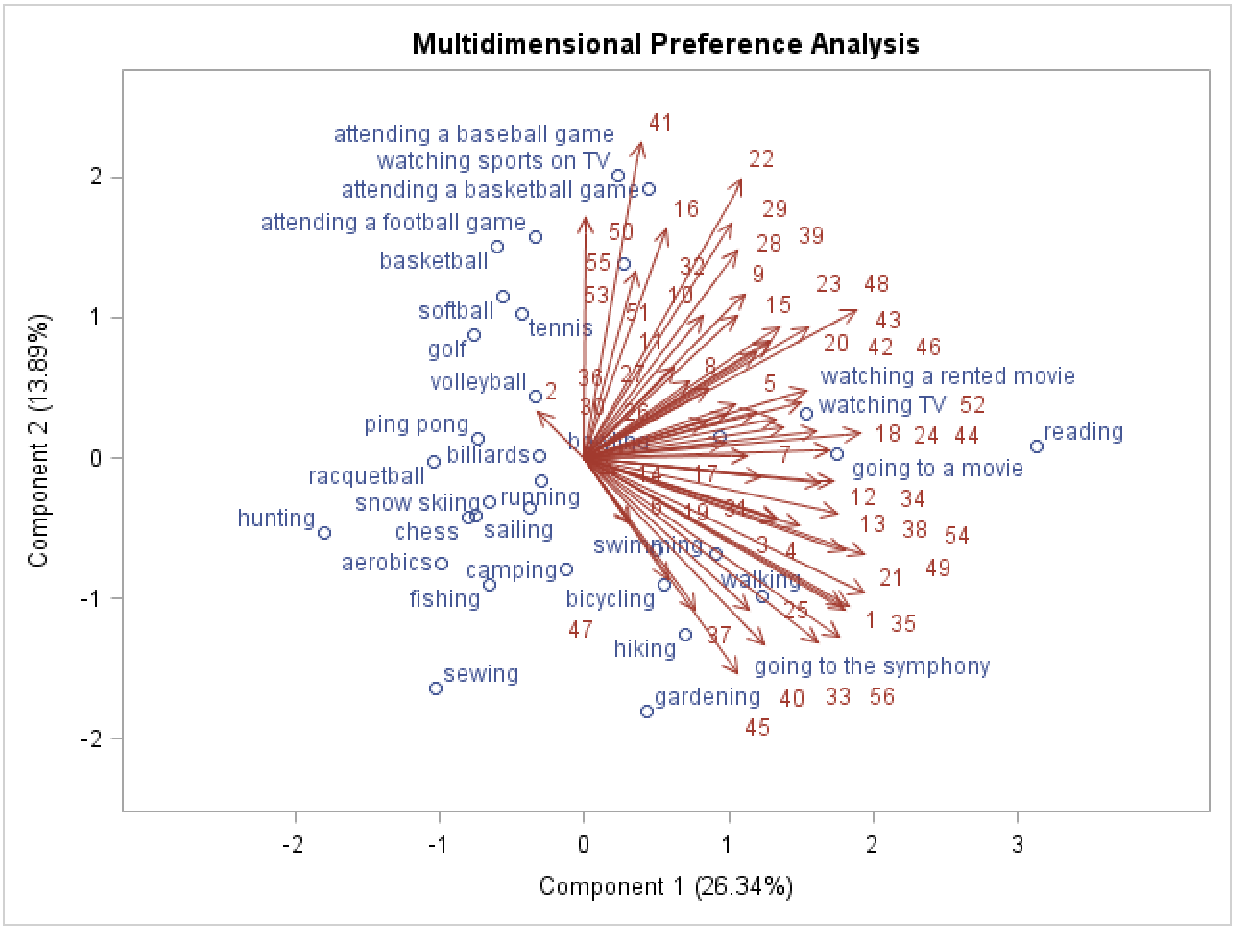


Figure 16. MDPREF Analysis from PROC PRINQUAL

The following statements create Figure 17:

```
data carpref2(%ren(judge,25)); set carpref; run;

ods graphics / reset=index imagename="obsodsmdp2";
proc prinqual data=carpref2 n=2 out=presults(drop='1'n-'25'n) replace mdpref;
   title 'Preference Ratings for Automobiles Manufactured in 1980';
   ods select mdprefplot;
   id model mpg reliable ride;
   transform identity('1'n-'25'n);
   run;

options validvarname=v7;
```

Like the previous example, the variables are renamed, and PROC PRINQUAL and the `mdpref` option automatically produce the MDPREF plot when ODS Graphics is enabled.

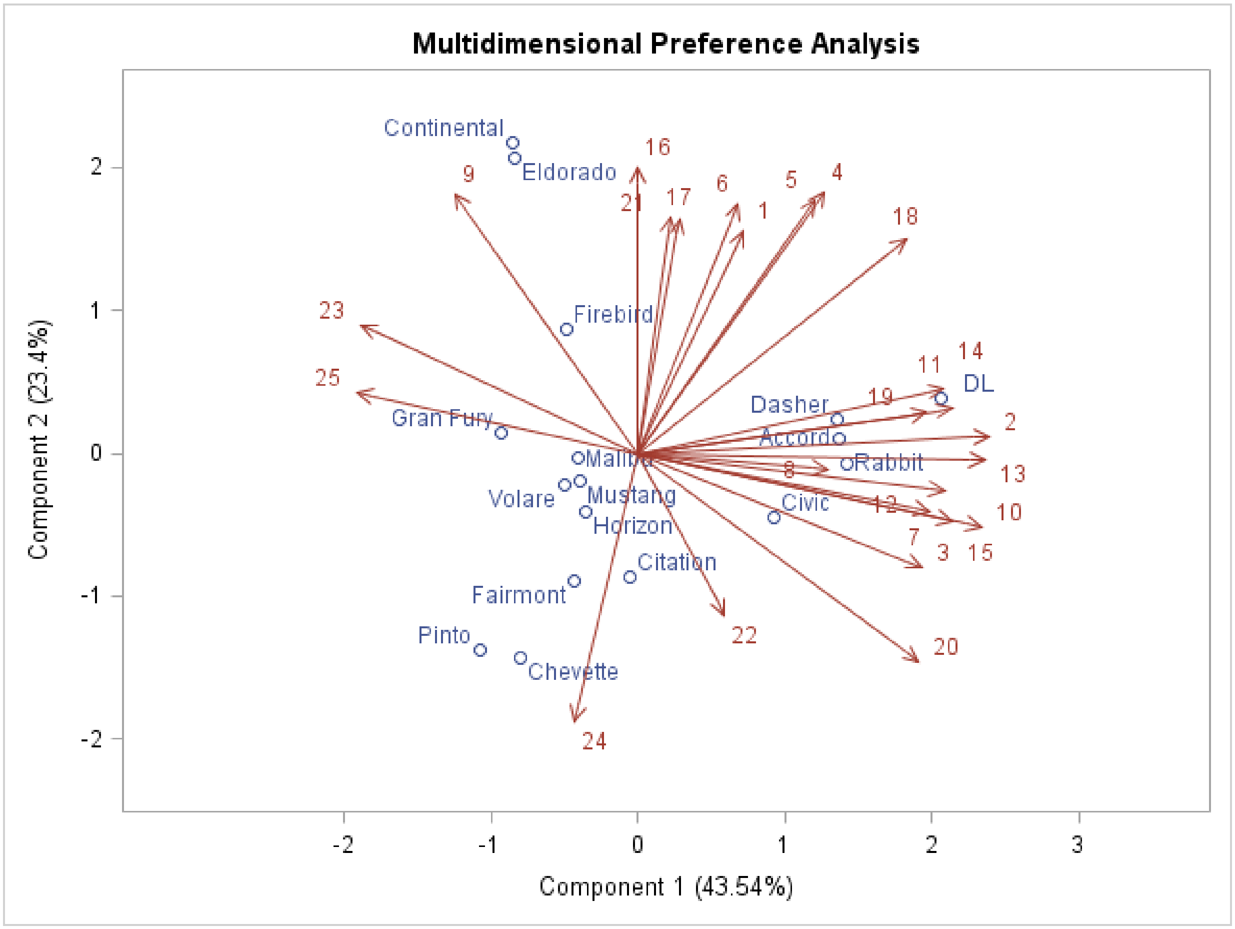


Figure 17. MDPREF Analysis from PROC PRINQUAL

The following statements create Figure 18, which is similar to Figure 7:

```
ods graphics / reset=index imagename="obsodsvec";
proc transreg data=presults tstandard=center coordinates;
   title2 'Preference Mapping, Vector Model';
   ods select prefmapvecplot;
   model identity(mpg reliable ride)=identity(prin1 prin2);
   id model;
   run;
```

When ODS Graphics is enabled, PROC TRANSREG and the `coordinates` option automatically produce a PREFMAP plot. Since the independent variable are designated as `identity` variables, the PREFMAP plot is a vector plot.

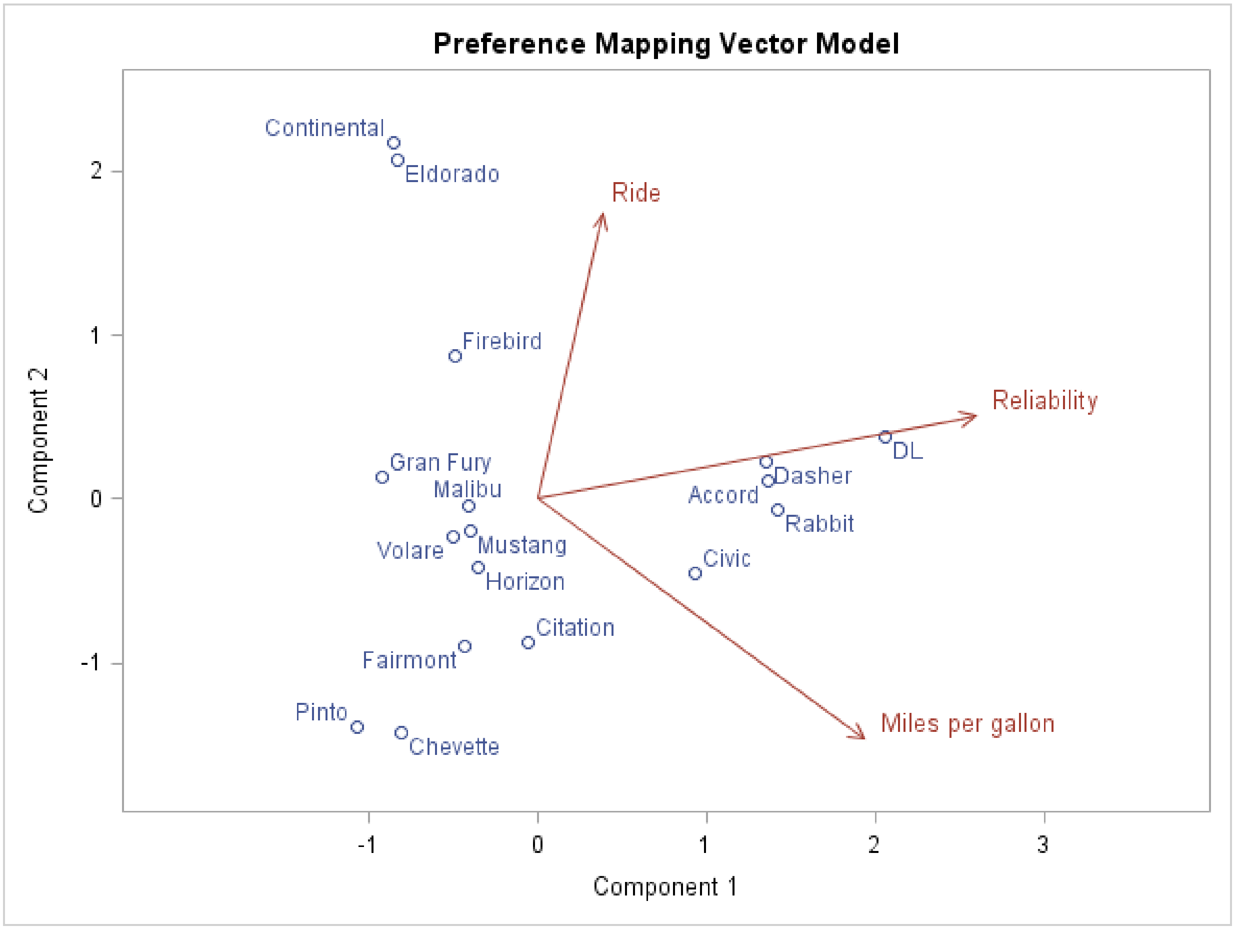


Figure 18. *PREFMAP Vector Model from PROC TRANSREG*

The following statements create Figure 19, which is similar to Figure 8:

```
ods graphics / reset=index imagename="obsodsidp";
proc transreg data=presults tstandard=center coordinates ;
   title2 'Preference Mapping, Ideal Point Model';
   ods select prefmapidealplot;
   model identity(mpg reliable ride)=point(prin1 prin2);
   id model;
   run;
```

When ODS Graphics is enabled, PROC TRANSREG and the `coordinates` option automatically produce a PREFMAP plot. Since the independent variable are designated as `point` variables, the PREFMAP plot is an ideal-point plot. ODS Graphics does not automatically draw the circles like the `%PlotIt` macro does.

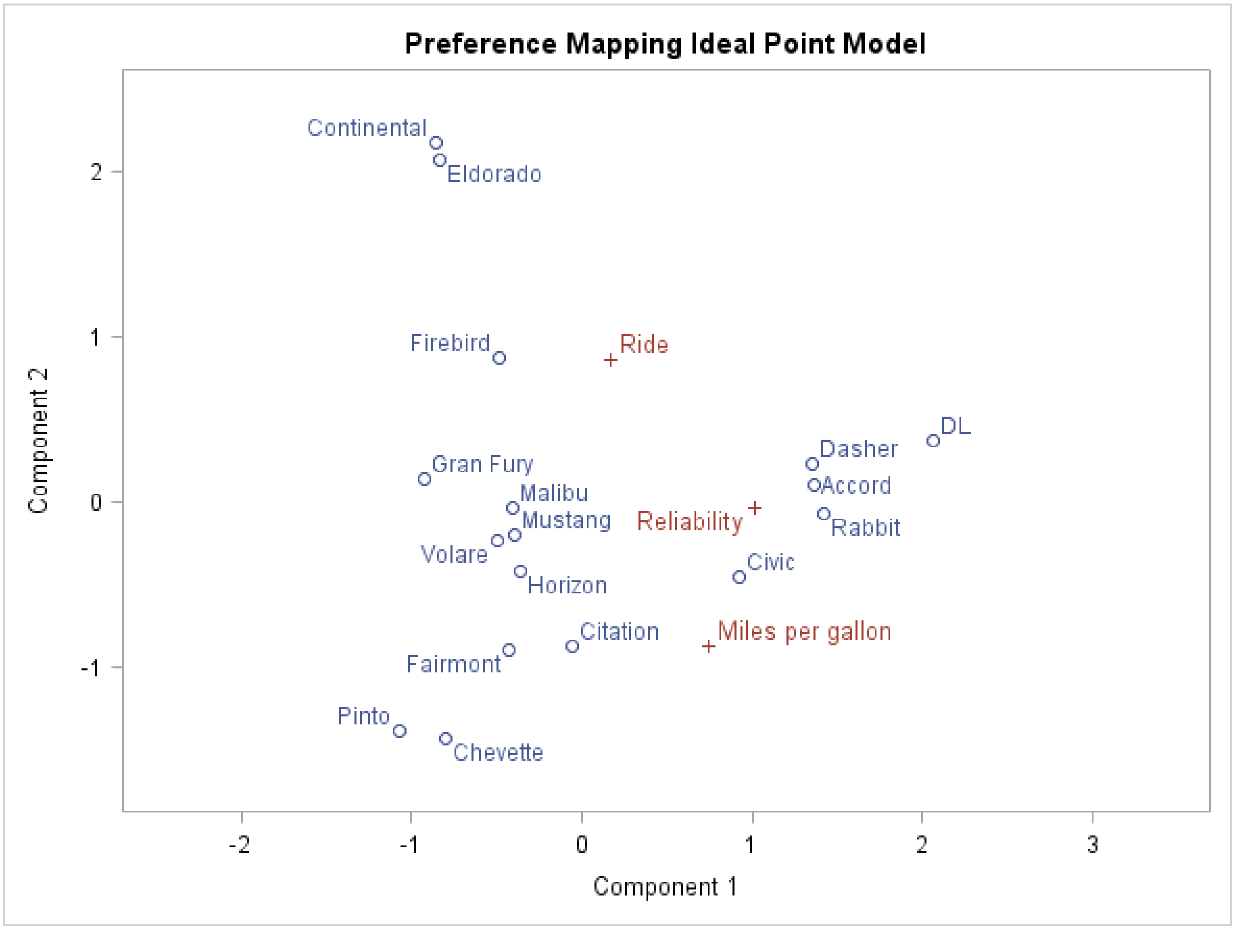


Figure 19. PREFMAP Ideal Point Model from PROC TRANSREG

The following statements create Figure 20, which is similar to Figures 9 and 10:

```
ods graphics / reset=index imagename="obsodsreg";
proc sgplot data=vital;
   title 'Crude Death Rate as a Function of Crude Birth Rate';
   pbspline y=death x=birth / datalabel=country nolegfit;
   run;
```

You can use PROC SGPLOT to directly create many types of scatter plots without having to make a template and use PROC SGRENDER.* This example uses the `pbspline` statement to plot a scatter plot of points and fit a nonlinear regression function using penalized B-splines. This statement automatically finds a smooth function based on an automatically-chosen smoothing parameter.

With PROC SGPLOT, the `title` statement provides that title that actually appears in the `png` file with the plot. In PROC SGRENDER, the `entrytitle` statement provides that title that actually appears in the `png` file with the plot. In regular, non-SG procedures, the title appears in the output but outside the plot, not as part of the plot. The plot titles are determined by the plot template.

*PROC TEMPLATE and PROC SGRENDER were used in a previous example because PROC SGPLOT does not currently support the continuous legend. Occasionally, you will need other capabilities not in PROC SGPLOT, and then you will have to use PROC TEMPLATE and PROC SGRENDER. For example, when you want to equate the axes in a plot, you will have to use PROC TEMPLATE and PROC SGRENDER.

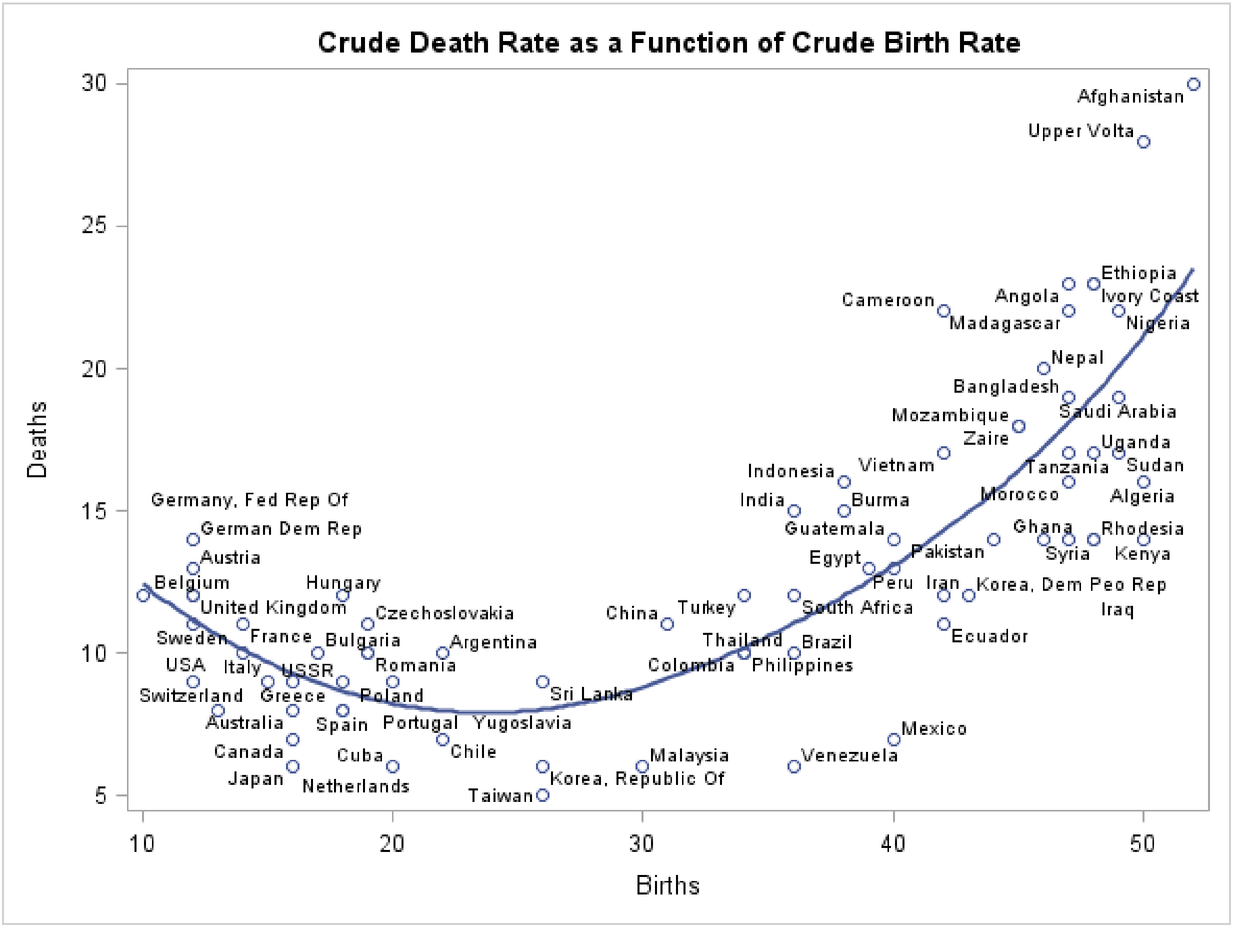


Figure 20. Spline Regression Fit from PROC SGPLOT

The following statements create Figure 21:

```
ods graphics / reset=index imagename="obsodstreg";
proc transreg data=vital;
   title 'Crude Death Rate as a Function of Crude Birth Rate';
   ods select fitplot;
   id country;
   model identity(death) = mspline(birth / nknots=9);
   run;
```

When ODS Graphics is enabled, PROC TRANSREG automatically produces a fit plot when it is appropriate. When a simple regression model is being fit, optionally with an independent variable transformation and up to one classification variable, a fit plot is automatically produced. In this example, a nonlinear but monotone (always goes up or stays flat, or always goes down or stays flat, but never goes both up and down) fit function is used. A fit plot with separate functions for each level of the classification variable could be produced by using a model like the following with PROC TRANSREG and ODS Graphics enabled:

```
model identity(death) = class(continent / zero=none) |
                        spline(birth / nknots=9);
```

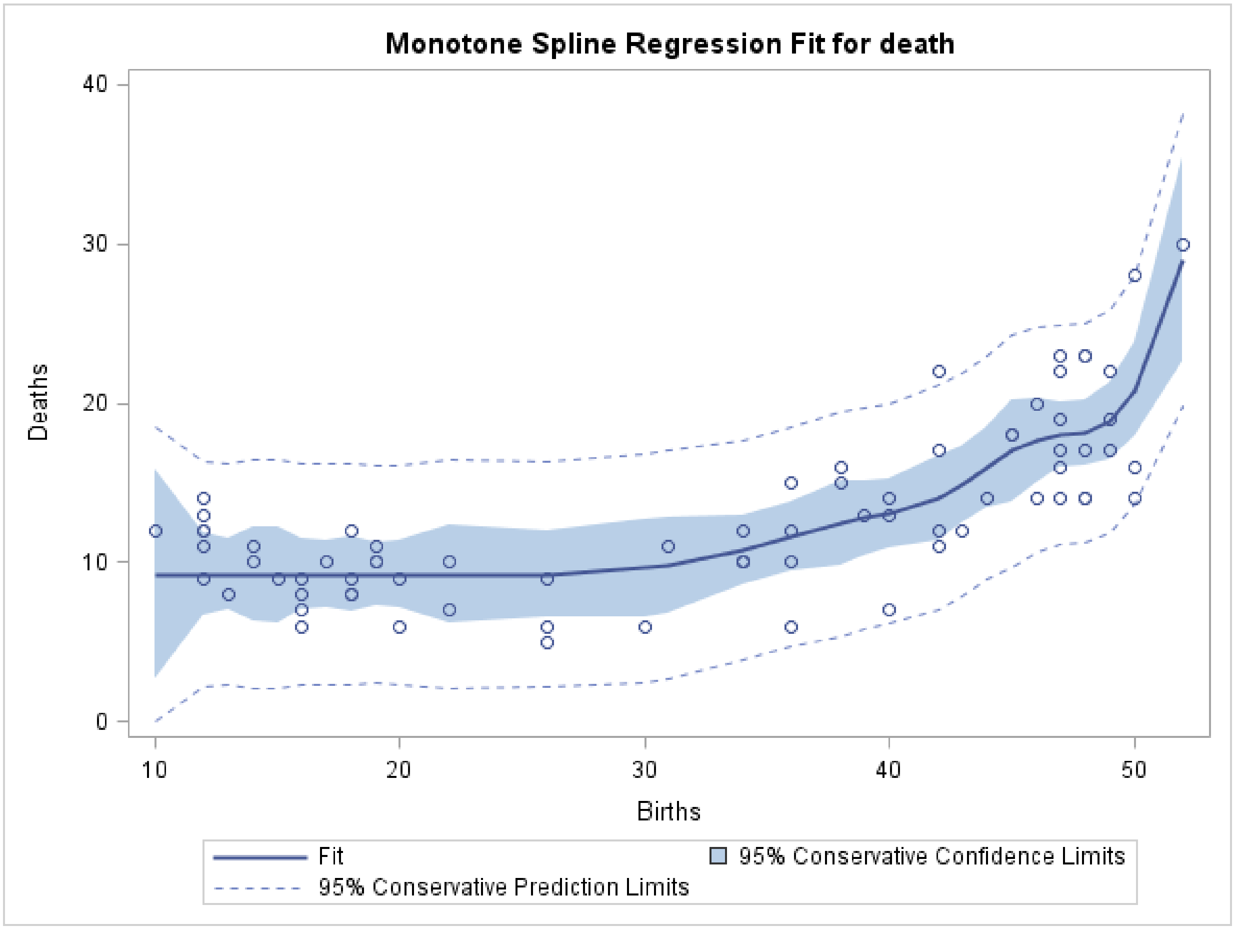


Figure 21. Monotone Spline Regression Fit from PROC TRANSREG

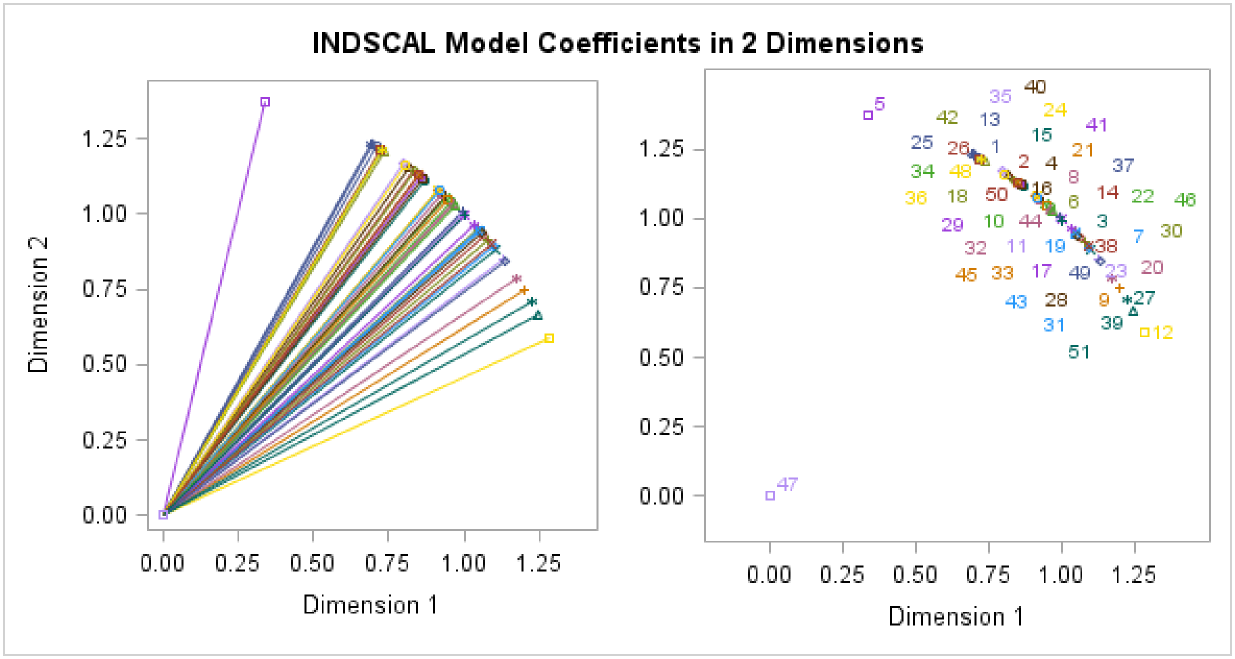


Figure 23. Coefficient Plot from PROC MDS

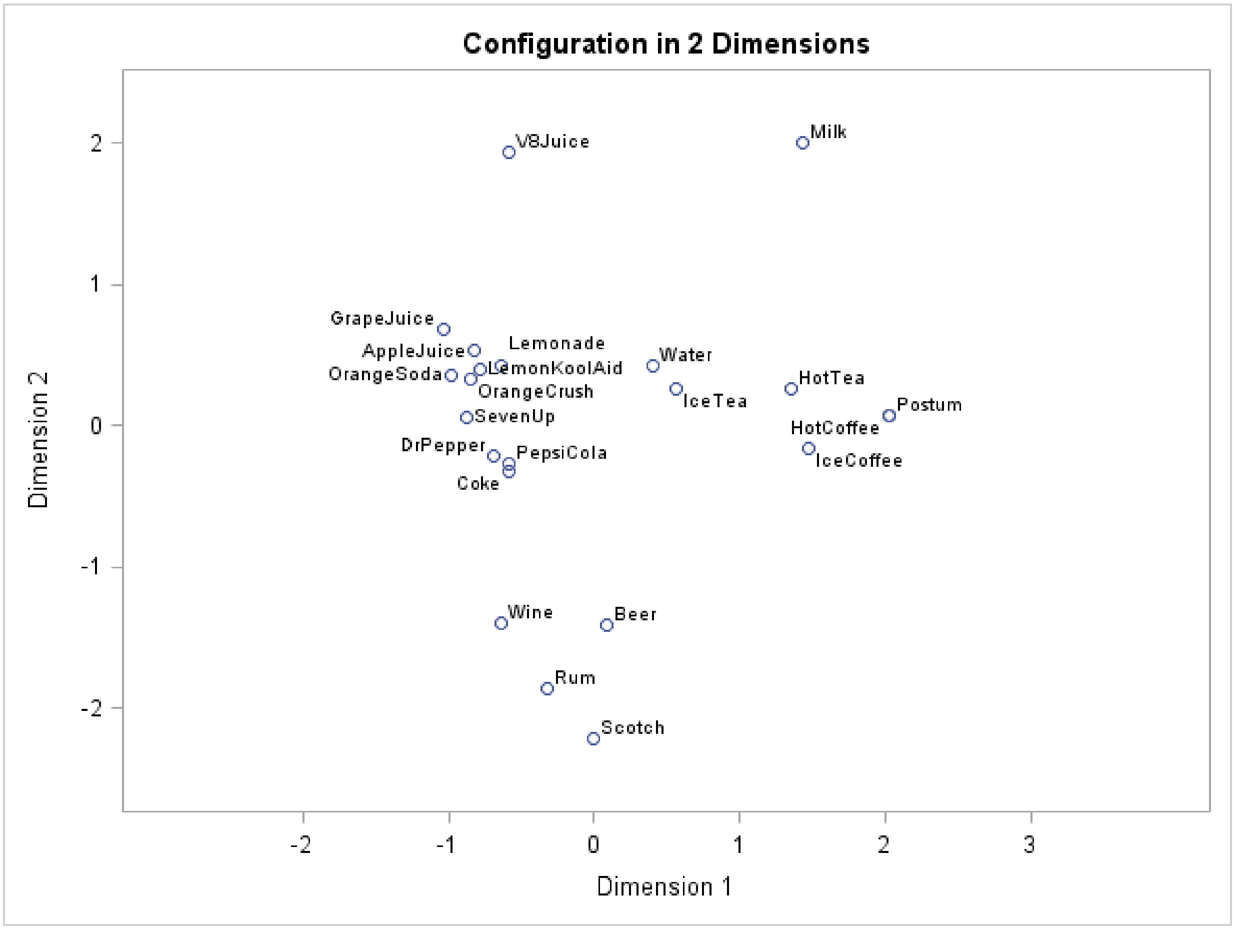


Figure 22. Configuration Plot from PROC MDS

The following statements create Figures 22, 23, and 24:

```
ods graphics / reset=index imagename="obsodsmds";
proc mds data=dissim level=interval model=indscal dimens=2 header;
   ods select configplot coefficientsplot fitplot;
   title 'Multidimensional Scaling of Beverages';
   run;

ods graphics off;
```

When ODS Graphics is enabled, PROC MDS automatically produces plots of the results of the MDS analysis. Figure 22 shows the configuration of points. Figure 23 shows the coefficients vectors for the INDSCAL model. The plot on the left of the panel shows the appropriate geometry with the coefficients displayed as vectors. The plot on the right, provides in some sense, a legend for the plot on the left. Just the vector end points are shown and they are labeled with subject ID information, which in this example is just the subject number. Figure 24 shows the fit of the data to the two-dimensional solution.

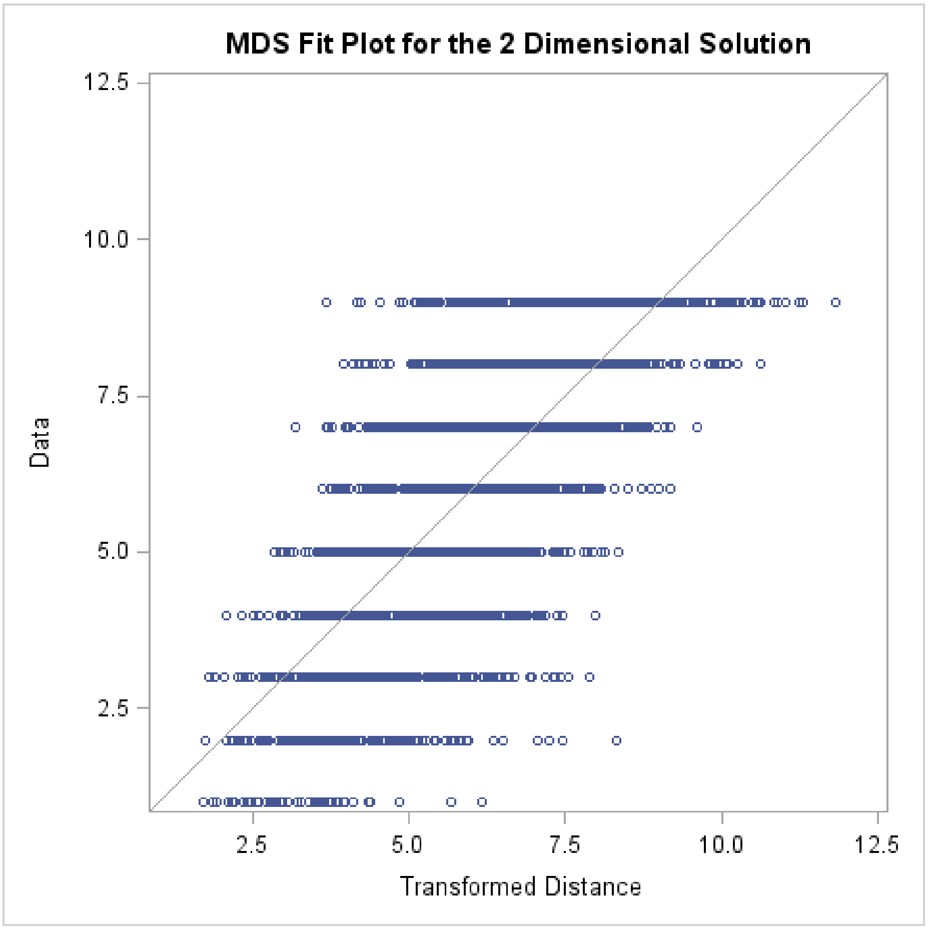


Figure 24. Fit Plot from PROC MDS

Since this step produces three plots, the file names are: `png/obsodsmds.png`, `png/obsodsmds1.png`, and `png/obsodsmds2.png`.

Graphical Methods for Marketing Research

Warren F. Kuhfeld

Abstract

Correspondence analysis, multiple correspondence analysis, preference mapping, and multidimensional preference analysis are descriptive statistical methods that generate graphical displays from data matrices. These methods are used by marketing researchers to investigate relationships among products and individual differences in preferences for those products. The end result is a two- or three-dimensional scatter plot that shows the most salient information in the data matrix. This chapter describes these methods, shows examples of the graphical displays, and discusses marketing research applications.[*]

Introduction

Correspondence analysis (CA), multiple correspondence analysis (MCA), preference mapping (PREFMAP), and multidimensional preference analysis (MDPREF) are descriptive statistical methods that generate graphical displays from data matrices. These methods are sometimes referred to as perceptual mapping methods. They simultaneously locate two or more sets of points in a single plot, and all emphasize presenting the geometry of the data. CA simultaneously displays in a scatter plot the row and column labels from a two-way contingency table or crosstabulation constructed from two categorical variables. MCA simultaneously displays in a scatter plot the category labels from more than two categorical variables. MDPREF displays both the row labels (products) and column labels (people) from a data matrix of continuous variables. PREFMAP shows rating scale data projected into a plot of row labels—for example, from an MDPREF analysis. These methods are used by marketing researchers to investigate relationships among products and individual differences in preferences for those products.

This chapter will only discuss these techniques as methods of generating two-dimensional scatter plots. However, three-dimensional and higher-dimensional results can also be generated and displayed with modern interactive graphics software and with scatter plot matrices.

[*]Copies of this chapter (MR-2010L), the other chapters, sample code, and all of the macros are available on the Web `http://support.sas.com/resources/papers/tnote/tnote_marketresearch.html`. This chapter is a revision of a paper that was published in the 1992 National Computer Graphics Association Conference Proceedings. For help, please contact SAS Technical Support. See page 25 for more information.

Methods

This section presents the algebra and example plots for MDPREF, PREFMAP, CA, and MCA. These methods are all similar in spirit to the biplot, which is discussed first to provide a foundation for the other methods.

The Biplot. A *biplot* (Gabriel 1981) simultaneously displays the rows and columns of a data matrix in a low-dimensional (typically two-dimensional) plot. The "bi" in "biplot" refers to the *joint* display of rows and columns, not to the dimensionality of the plot. Consider an $(n \times m)$ data matrix $\mathbf{Y}$, an $(n \times q)$ matrix $\mathbf{A}$ with row vectors $\mathbf{a}_1'$, $\mathbf{a}_2'$, ..., $\mathbf{a}_n'$, and an $(m \times q)$ matrix $\mathbf{B}$ with row vectors $\mathbf{b}_1'$, $\mathbf{b}_2'$, ..., $\mathbf{b}_m'$. The n rows of $\mathbf{A}$ correspond to the rows of $\mathbf{Y}$, and the m columns of $\mathbf{B}'$ correspond to the columns of $\mathbf{Y}$. The rank of $\mathbf{Y}$ is $q \leq \text{MIN}(n, m)$. $\mathbf{A}$ and $\mathbf{B}$ are chosen such that $y_{ij} = \mathbf{a}_i'\mathbf{b}_j$. If $q = 2$ and the rows of $\mathbf{A}$ and $\mathbf{B}$ are plotted in a two-dimensional scatter plot, the scalar product of the coordinates $\mathbf{a}_i'$ and $\mathbf{b}_j'$ *exactly* equals the data value y_{ij}. This kind of scatter plot is a biplot; it geometrically shows the algebraic relationship $\mathbf{AB}' = \mathbf{Y}$. Typically, the row coordinates are plotted as points, and the column coordinates are plotted as vectors.

When $q > 2$ and two dimensions are plotted, then $\mathbf{a}_i'\mathbf{b}_j$ is *approximately* equal to y_{ij}, and the display is an *approximate biplot*.[†] The approximate biplot geometrically shows the algebraic relationship $\mathbf{AB}' \approx \mathbf{Y}$. The best values for $\mathbf{A}$ and $\mathbf{B}$, in terms of minimum squared error in approximating $\mathbf{Y}$, are found using a singular value decomposition (SVD),[‡] $\mathbf{Y} = \mathbf{AB}' = \mathbf{UDV}'$, where $\mathbf{D}$ is a $(q \times q)$ diagonal matrix and $\mathbf{U}'\mathbf{U} = \mathbf{V}'\mathbf{V} = \mathbf{I}_q$, a $(q \times q)$ identity matrix. Solutions for $\mathbf{A}$ and $\mathbf{B}$ include $\mathbf{A} = \mathbf{U}$ and $\mathbf{B} = \mathbf{VD}$, or $\mathbf{A} = \mathbf{UD}$ and $\mathbf{B} = \mathbf{V}$, or more generally $\mathbf{A} = \mathbf{UD}^r$ and $\mathbf{B} = \mathbf{VD}^{(1-r)}$, for $0 \leq r \leq 1$. See Gabriel (1981) for more information about the biplot.

Multidimensional Preference Analysis. Multidimensional Preference Analysis (Carroll 1972) or MDPREF is a biplot analysis for preference data. Data are collected by asking respondents to rate their preference for a set of objects. Typically in marketing research, the objects are products—the client's products and the competitors'. Questions that can be addressed with MDPREF analyses include: Who are my customers? Who else should be my customers? Who are my competitors' customers? Where is my product positioned relative to my competitors' products? What new products should I create? What audience should I target for my new products?

For example, consumers can be asked to rate their preference for a group of automobiles on a 0 to 9 scale, where 0 means no preference and 9 means high preference. $\mathbf{Y}$ is an $(n \times m)$ matrix that contains ratings of the n products by the m consumers. The data are stored as the transpose of the typical data matrix, since the columns are the people. The goal is to produce a plot with the cars represented as points and the consumers represented as vectors. Each person's vector points in *approximately* the direction of the cars that the person most preferred and away from the cars that are least preferred.

Figure 1 contains an example in which 25 consumers rated their preference for 17 new (at the time) 1980 automobiles. This plot is based on a principal component model. It differs from a proper biplot of $\mathbf{Y}$ due to scaling factors. In principal components, the columns in data matrix $\mathbf{Y}$ are standardized to mean zero and variance one. The SVD is $\mathbf{Y} = \mathbf{UDV}'$, and the principal component model is $\mathbf{Y} = ((n-1)^{1/2}\mathbf{U})\,((n-1)^{-1/2}\mathbf{D})\,(\mathbf{V}')$. The standardized principal component scores matrix, $\mathbf{A} = (n-1)^{1/2}\mathbf{U}$, and the component structure matrix, $(n-1)^{-1/2}\mathbf{DV}'$, are plotted. The advantage of creating a biplot based on $(n-1)^{1/2}\mathbf{U}$ and $(n-1)^{-1/2}\mathbf{DV}'$ instead of $\mathbf{U}$ and $\mathbf{DV}'$ is that the coordinates

[†]In practice, the term biplot is sometimes used without qualification to refer to an approximate biplot.

[‡]SVD is sometimes referred to in the psychometric literature as an Eckart-Young (1936) decomposition.

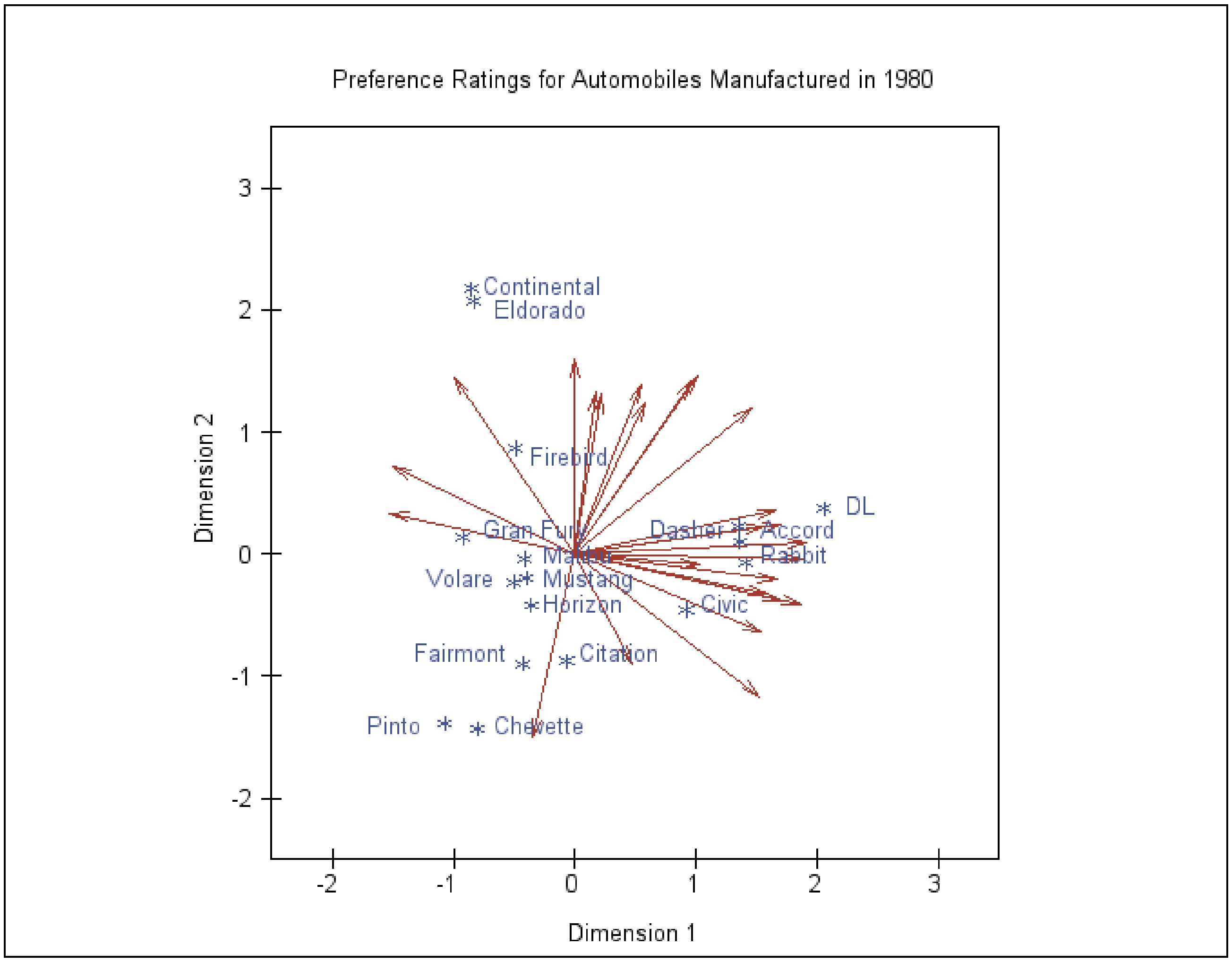


Figure 1. Multidimensional Preference Analysis

do not get smaller as sample size increases. The fit, or proportion of the variance in the data accounted for by the first two dimensions, is the sum of squares of the first two elements of $(n-1)^{-1/2}\mathbf{D}$ divided by the sum of squares of all of the elements of $(n-1)^{-1/2}\mathbf{D}$.

The dimensions of the MDPREF biplot are the first two principal components. The first principal component represents the information that is most salient to the preference judgments. At one end of the plot of the first principal component are the most preferred automobiles; the least preferred automobiles are at the other end of the plot. The second principal component represents the direction that is most salient to the preference judgments that is orthogonal to the first principal component. The automobile point coordinates are the scores of the automobile on the first two principal components. The judge vectors point in *approximately* the direction of judges most preferred cars, with preference increasing as the vector moves from the origin.

Let $\mathbf{a}_i'$ be row i of $\mathbf{A} = (n-1)^{1/2}\mathbf{U}$, $\mathbf{b}_j'$ be row j of $\mathbf{B} = (n-1)^{-1/2}\mathbf{VD}$, $\|\mathbf{a}_i\|$ be the length of $\mathbf{a}_i$, $\|\mathbf{b}_j\|$ be the length of $\mathbf{b}_j$, and θ be the angle between the vectors $\mathbf{a}_i$ and $\mathbf{b}_j$. The predicted degree of (scaled) preference that an individual judge has for an automobile is $\mathbf{a}_i'\mathbf{b}_j = \|\mathbf{a}_i\|\,\|\mathbf{b}_j\|cos\theta$. Each car point can be orthogonally projected onto each judge's vector. The projection of the *ith* car on the *jth* judge vector is $\mathbf{b}_j((\mathbf{a}_i'\mathbf{b}_j)/(\mathbf{b}_j'\mathbf{b}_j))$, and the length of this projection is $\|\mathbf{a}_i\|cos\theta$. The automobile that projects farthest along a judge vector has the highest predicted preference. The length of this projection, $\|\mathbf{a}_i\| cos\theta$, differs from the predicted preference, $\|\mathbf{a}_i\| \, \|\mathbf{b}_j\| cos\theta$, only by $\|\mathbf{b}_j\|$, which is constant within each judge. Since the goal is to look at projections of points onto the vectors, the absolute length of a judge's vector is unimportant. The relative lengths of the vectors indicate fit, with longer vectors indicating better fit. The coordinates for the endpoints of the vectors were multiplied by 2.5 to extend the vectors and create a better graphical display. The direction of the preference scale is important. The vectors point in the direction of increasing values of the data values. If the data had been ranks, with 1 the most preferred and n the least preferred, then the vectors would point in the direction of the least preferred automobiles.

The people in the top left portion of the plot most prefer the large American cars. Other people, with vectors pointing up and nearly vertical, also show this pattern of preference. There is a large cluster of people who prefer the Japanese and European cars. A few people, most notably the person whose vector passes through the "e" in "Chevette", prefer the small and inexpensive American Cars. There are no vectors pointing through the bottom left portion of the of the plot, which suggests that the smaller American cars are generally not preferred by anyone within this group.

The first dimension, which is a measure of overall evaluation, discriminates between the American cars on the left and the Japanese and European cars on the right. The second dimension seems to reflect the sizes of the automobiles. Some cars have a similar pattern of preference, most notably Continental and Eldorado, which share a symbol in the plot. Marketers of Continental or Eldorado may want to try to distinguish their car from the competition. Dasher, Accord, and Rabbit were rated similarly, as were Malibu, Mustang, Volare, and Horizon.

This 1980 example is quite prophetic even though it is based on a small nonrandom sample. Very few vectors point toward the smaller American cars, and Mustang is the only one of them that is still being made. Many vectors are pointing toward the European and Japanese cars, and they are still doing quite well in the market place. Many vectors are pointing in the one to two o'clock range where there are no cars in the plot. One can speculate that these people would prefer Japanese and European luxury cars such as Accura, Lexus, Infinity, BMW, and Mercedes.

Preference Mapping. Preference mapping[*] (Carroll 1972) or PREFMAP plots resemble biplots, but are based on a different model. The goal in PREFMAP is to take a set of coordinates for a set of objects, such as the MDPREF car coordinates in example in Figure 1, and project in external information that can aid in interpreting the configuration of points. Questions that can be addressed with PREFMAP analyses include: Where is my product positioned relative to my competitors' products? Why is my product positioned there? How can I reposition my existing products? What new products should I create?

The Preference Mapping Vector Model. Figure 2 contains an example in which three attribute variables (ride, reliability, and miles per gallon) are displayed in the plot of the first two principal components of the car preference data. Each of the automobiles was rated on these three dimensions on a 1 to 5 scale, where 1 is poor and 5 is good. Figure 2 is based on the simplest version of PREFMAP—the *vector model.* The vector model assumes that some is good and more is *always* better. This model is appropriate for miles per gallon and reliability—the more miles a motorist can travel without refueling or breaking down, the better. The end points for the attribute vectors are obtained by projecting the attribute variables into the car space. If the attribute ratings are stored in matrix $\mathbf{R}$, then the coordinates for the end points are in the matrix $\boldsymbol{\beta}$ from the multivariate linear regression

[*]Preference mapping is sometimes referred to as external unfolding.

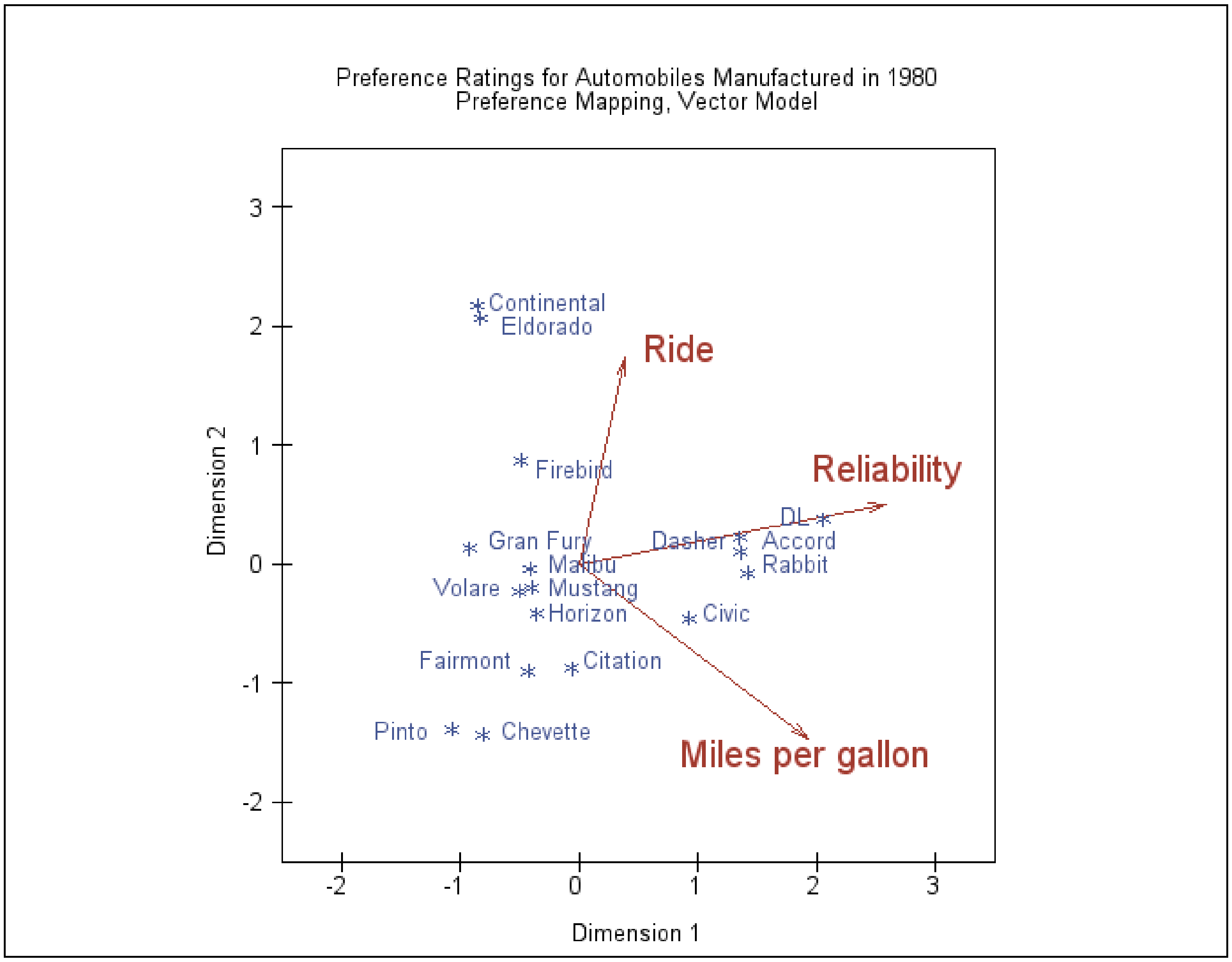


Figure 2. Preference Mapping, Vector Model

model $\mathbf{R} = \mathbf{A}\boldsymbol{\beta} + \boldsymbol{\epsilon}$. $\mathbf{A}$ is the matrix of standardized principal component scores, or $\mathbf{A}$ could be the coordinates from a multidimensional scaling analysis. The relative lengths of the vectors indicate fit, which is given by the R square. As with MDPREF, the lengths of all vectors can be scaled by the same constant to make a better graphical display.

PREFMAP analyses can help in the interpretation of principal component, multidimensional scaling, and MDPREF analyses by projecting in external information that helps explain the configuration. Orthogonal projections of the product points on an attribute vector give an *approximate* ordering of the products on the attribute. The ride vector points almost straight up showing that the larger cars, such as the Eldorado and Continental, have the best ride. In Figure 1, it was shown that most people preferred the DL, Japanese cars, and larger American cars. Figure 2 shows that the DL and Japanese cars were rated as the most reliable and have the best fuel economy. The small American cars are not rated highly on any of the three dimensions, although some are on the positive end of miles per gallon.

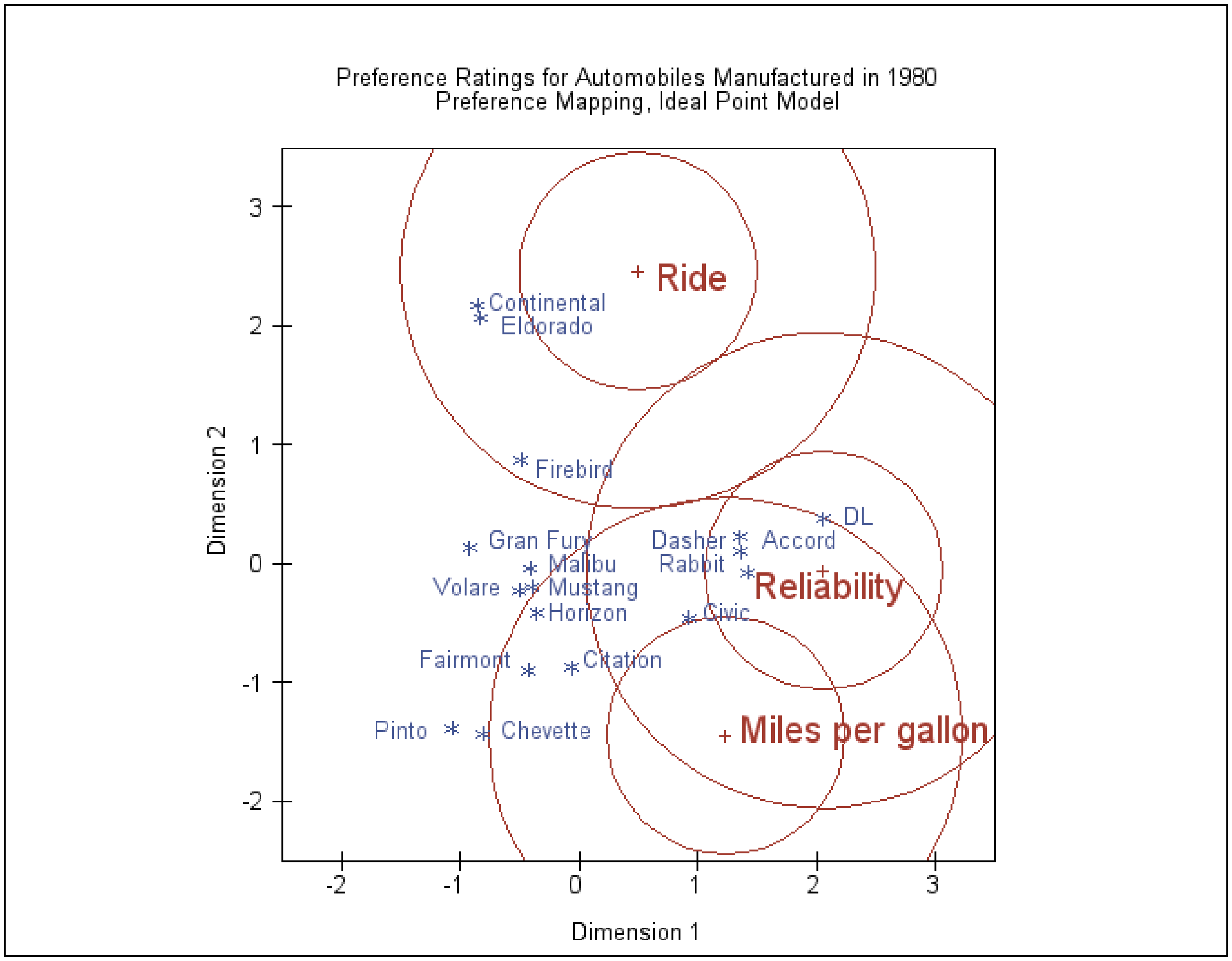


Figure 3. Preference Mapping, Ideal Point Model

The Preference Mapping Ideal Point Model. The *ideal point* model differs from the vector model in that the ideal point model does not assume that more is better, *ad infinitum.* Consider the sugar content of cake. There is an ideal amount of sugar that cake should contain—not enough sugar is not good, and too much sugar is also not good. In the current example, the ideal number of miles per gallon and the ideal reliability are unachievable. It makes sense to consider a vector model, because the ideal point is infinitely far away. This argument is less compelling for ride; the point for a car with smooth, quiet ride may not be infinitely far away. Figure 3 shows the results of fitting an ideal point model for the three attributes. In the vector model, results are interpreted by orthogonally projecting the car points on the attribute vectors. In the ideal point model, Euclidean distances between car points and ideal points are compared. Eldorado and Continental have the best predicted ride, because they are closest to the ride ideal point. The concentric circles drawn around the ideal points help to show distances between the cars and the ideal points. The numbers of circles and their radii are arbitrary. The overall interpretations of Figures 2 and 3 are the same. All three ideal points are at the edge of the car points, which suggests the simpler vector model is sufficient for these data.

The ideal point model is fit with a multiple regression model and some pre- and post-processing. First the $\mathbf{A}$ matrix is augmented by a variable that is the sum of squares of columns of $\mathbf{A}$ creating $\mathbf{A}^*$. Then solve for $\boldsymbol{\beta}$ from $\mathbf{R} = \mathbf{A}^*\boldsymbol{\beta} + \boldsymbol{\epsilon}$. For a two-dimensional scatter plot, the ideal point coordinates are given by dividing each coefficient for the two axes by the coefficient for the sum-of-squares variable, then multiplying the resulting values by -0.5. The coordinates are $-0.5\,\boldsymbol{\beta}\,\mathrm{diag}(\boldsymbol{\beta}_3)^{-1}$, where $\mathrm{diag}(\boldsymbol{\beta}_3)$ is a diagonal matrix constructed from the third row of $\boldsymbol{\beta}$. This is a constrained response-surface model. The fit is given by the R square. See Carroll (1972) for the justification for the formula.

The results in Figure 3 were modified from the raw results to eliminate *anti-ideal points*. The ideal point model is a distance model. The rating data are interpreted as distances between attribute ideal points and the products. In this example, each of the automobiles was rated on these three dimensions, on a 1 to 5 scale, where 1 is poor and 5 is good. The data are the reverse of what they should be—a ride rating of 1 should mean this car is similar to a car with a good ride, and a rating of 5 should mean this car is different from a car with a good ride. So the raw coordinates must be multiplied by -1 to get ideal points. Even if the scoring had been reversed, anti-ideal points can occur. If the coefficient for the sum-of-squares variable is negative, the point is an anti-ideal point. In this example, there is the possibility of *anti-anti-ideal points*. When the coefficient for the sum-of-squares variable is negative, the two multiplications by -1 cancel, and the coordinates are ideal points. When the coefficient for the sum-of-squares variable is positive, the coordinates are multiplied by -1 to get an ideal point.

Other PREFMAP Models. The ideal point model presented here is based on an ordinary Euclidean distance model. All points falling on a circle centered around an ideal point are an equal distance from the ideal point. Two more PREFMAP models are sometimes used. The more general models allow for differential weighting of the axes and rotations, so ellipses, not circles, show equal weighted distances. All three ideal point models are response surface models. See Carroll (1972) for more information.

Correspondence Analysis. Correspondence analysis (CA) is a weighted SVD of a contingency table. It is used to find a low-dimensional graphical representation of the association between rows and columns of a table. Each row and column is represented by a point in a Euclidean space determined from cell frequencies. Like MDPREF, CA is based on a singular value decomposition, but ordinary SVD of a contingency table does not portray a desirable geometry.

Questions that can be addressed with CA and MCA include: Who are my customers? Who else should be my customers? Who are my competitors' customers? Where is my product positioned relative to my competitors' products? Why is my product positioned there? How can I reposition my existing products? What new products should I create? What audience should I target for my new products?

CA is a popular data analysis method in France and Japan. In France, CA was developed under the strong influence of Jean-Paul Benzécri; in Japan, under Chikio Hayashi. CA is described in Lebart, Morineau, and Warwick (1984); Greenacre (1984); Nishisato (1980); Tenenhaus and Young (1985); Gifi (1990); Greenacre and Hastie (1987); and many other sources. Hoffman and Franke (1986) provide a good introductory treatment using examples from marketing research.

Simple CA. This section is primarily based on the theory of CA found in Greenacre (1984). Let $\mathbf{N}$ be an $(n_r \times n_c)$ contingency table of rank $q \leq MIN(n_r, n_c)$. Let $\mathbf{1}$ be a vector of ones of the appropriate order, $\mathbf{I}$ be an identity matrix, and diag() be a matrix-valued function that creates a diagonal matrix from a vector. Let $f = \mathbf{1}'\mathbf{N}\mathbf{1}$, $\mathbf{P} = (1/f)\mathbf{N}$, $\mathbf{r} = \mathbf{P}\mathbf{1}$, $\mathbf{c} = \mathbf{P}'\mathbf{1}$, $\mathbf{D_r} = \mathrm{diag}(\mathbf{r})$, and $\mathbf{D_c} = \mathrm{diag}(\mathbf{c})$. The scalar f is the sum of all elements in $\mathbf{N}$. $\mathbf{P}$ is a matrix of relative frequencies. The vector $\mathbf{r}$ contains row marginal proportions or row *masses*. The vector $\mathbf{c}$ contains column marginal proportions or column masses. $\mathbf{D_r}$ and $\mathbf{D_c}$ are diagonal matrices of marginals. The coordinates of the CA are based on the generalized singular value decomposition of $\mathbf{P}$, $\mathbf{P} = \mathbf{A}\mathbf{D_u}\mathbf{B}'$, where $\mathbf{A}'\mathbf{D_r}^{-1}\mathbf{A} = \mathbf{B}'\mathbf{D_c}^{-1}\mathbf{B} = \mathbf{I}$. $\mathbf{A}$

is an $(n_r \times q)$ matrix of left generalized singular vectors, $\mathbf{D_u}$ is a $(q \times q)$ diagonal matrix of singular values, and $\mathbf{B}$ is an $(n_c \times q)$ matrix of right generalized singular vectors. The first (trivial) column of $\mathbf{A}$ and $\mathbf{B}$ and the first singular value in $\mathbf{D_u}$ are discarded before any results are displayed. This step centers the table and is analogous to centering the data in ordinary principal component analysis. In practice, this centering is done by subtracting ordinary chi-square expected values from $\mathbf{P}$ before the SVD. The columns of $\mathbf{A}$ and $\mathbf{B}$ define the principal axes of the column and row point clouds, respectively. The fit, or proportion of the *inertia* (analogous to variance) in the data accounted for by the first two dimensions, is the sum of squares of the first two singular values, divided by the sum of squares of all of the singular values. Three sets of coordinates are typically available from CA, one based on rows, one based on columns, and the usual set is based on both rows and columns.

The *row profile* (conditional probability) matrix is defined as $\mathbf{R} = \mathbf{D_r}^{-1}\mathbf{P} = \mathbf{D_r}^{-1}\mathbf{A}\mathbf{D_u}\mathbf{B}'$. Each (i, j) element of $\mathbf{R}$ contains the observed probability of being in column j given membership in row i. The values in each row of $\mathbf{R}$ sum to one. The row coordinates, $\mathbf{D_r}^{-1}\mathbf{A}\mathbf{D_u}$, and column coordinates, $\mathbf{D_c}^{-1}\mathbf{B}$, provide a CA based on the row profile matrix. The *principal* row coordinates, $\mathbf{D_r}^{-1}\mathbf{A}\mathbf{D_u}$, and *standard* column coordinates, $\mathbf{D_c}^{-1}\mathbf{B}$, provide a decomposition of $\mathbf{D_r}^{-1}\mathbf{A}\mathbf{D_u}\mathbf{B}'\mathbf{D_c}^{-1} = \mathbf{D_r}^{-1}\mathbf{P}\mathbf{D_c}^{-1} = \mathbf{R}\mathbf{D_c}^{-1}$. Since $\mathbf{D_r}^{-1}\mathbf{A}\mathbf{D_u} = \mathbf{R}\mathbf{D_c}^{-1}\mathbf{B}$, the row coordinates are weighted centroids of the column coordinates. Each column point, with coordinates scaled to standard coordinates, defines a vertex in $(n_c - 1)$-dimensional space. All of the principal row coordinates are located in the space defined by the standard column coordinates. Distances among row points have meaning, but distances among column points and distances between row and column points are not interpretable.

The formulas for the analysis of the *column profile* matrix can easily be derived by applying the row profile formulas to the transpose of $\mathbf{P}$. The principal column coordinates $\mathbf{D_c}^{-1}\mathbf{B}\mathbf{D_u}$ are weighted centroids of the standard row coordinates $\mathbf{D_r}^{-1}\mathbf{A}$. Each row point, with coordinates scaled to standard coordinates, defines a vertex in $(n_r - 1)$-dimensional space. All of the principal column coordinates are located in the space defined by the standard row coordinates. Distances among column points have meaning, but distances among row points and distances between row and column points are not interpretable.

The usual sets of coordinates[*] are given by $\mathbf{D_r}^{-1}\mathbf{A}\mathbf{D_u}$ and $\mathbf{D_c}^{-1}\mathbf{B}\mathbf{D_u}$. One advantage of using these coordinates is that both sets are postmultiplied by the diagonal matrix $\mathbf{D_u}$, whose diagonal values are all less than or equal to one. When $\mathbf{D_u}$ is a part of the definition of only one set of coordinates, that set forms a tight cluster near the centroid, whereas the other set of points is more widely dispersed. Including $\mathbf{D_u}$ in both sets makes a better graphical display. However, care must be taken in interpreting such a plot. No correct interpretation of distances between row points and column points can be made. Less specific statements, such as "two points are on the same side of the plot" have meaning.

Another property of this choice of coordinates concerns the geometry of distances between points within each set. Distances between row (or column) profiles are computed using a *chi-square metric*. The rationale for computing distances between row profiles using the non-Euclidean chi-square metric is as follows. Each row of the contingency table may be viewed as a realization of a multinomial distribution conditional on its row marginal frequency. The null hypothesis of row and column independence is equivalent to the hypothesis of homogeneity of the row profiles. A significant chi-square statistic is geometrically interpreted as a significant deviation of the row profiles from their centroid, $\mathbf{c}'$. The chi-square metric is the Mahalanobis metric between row profiles based on their estimated covariance matrix under the homogeneity assumption (Greenacre and Hastie 1987). A parallel argument can be made for the column profiles.

[*]This set is often referred to as the French standardization due to its popularity in France.

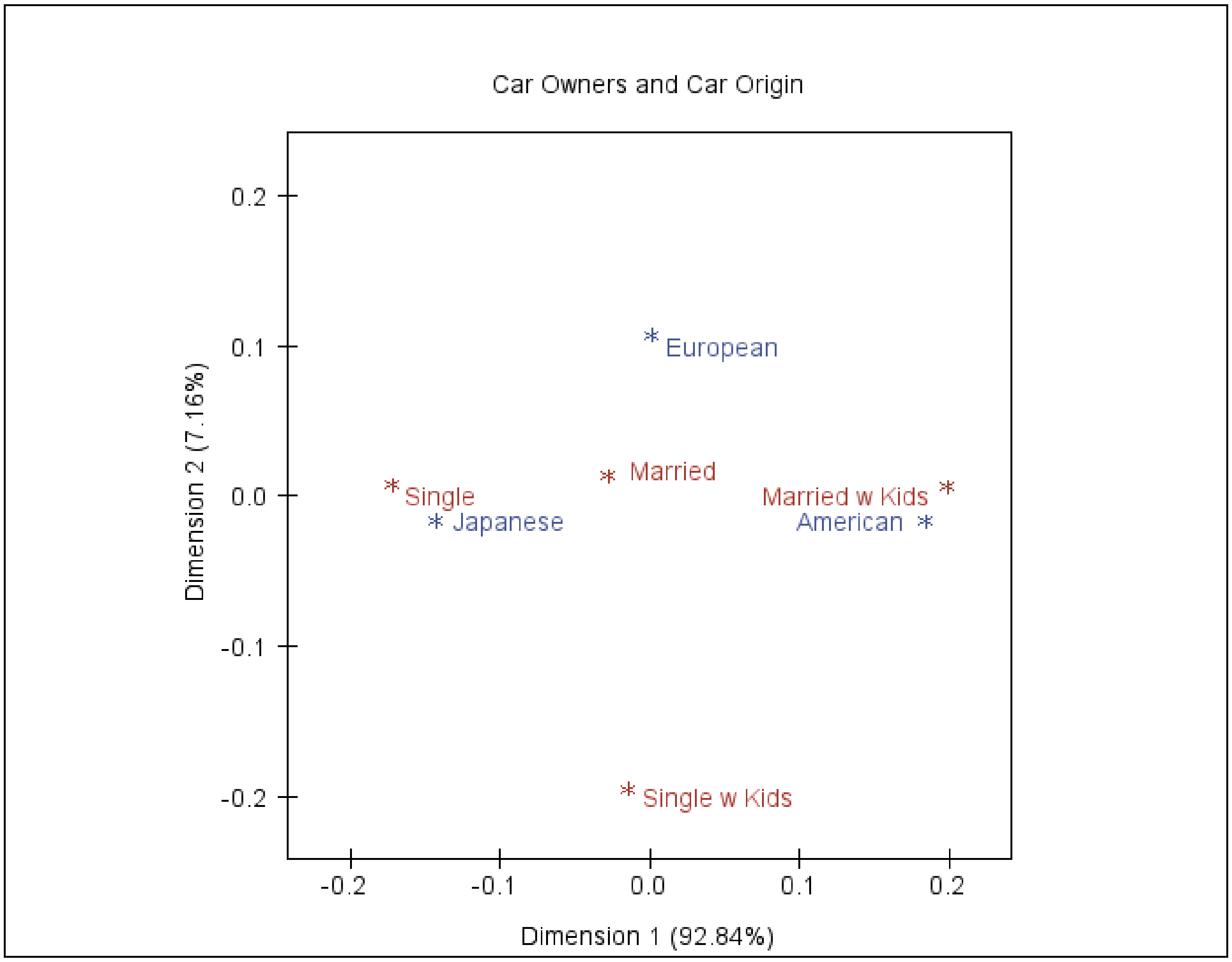


Figure 4. Simple Correspondence Analysis

The row coordinates are $\mathbf{D_r}^{-1}\mathbf{AD_u} = \mathbf{D_r}^{-1}\mathbf{AD_uB'D_c}^{-1}\mathbf{B} = (\mathbf{D_r}^{-1}\mathbf{P})(\mathbf{D_c}^{-1/2})(\mathbf{D_c}^{-1/2}\mathbf{B})$. They are row profiles $\mathbf{D_r}^{-1}\mathbf{P}$ rescaled by $\mathbf{D_c}^{-1/2}$ (rescaled so that distances between profiles are transformed from a chi-square metric to a Euclidean metric), then orthogonally rotated with $\mathbf{D_c}^{-1/2}\mathbf{B}$ to a principal axes orientation. Similarly, the column coordinates are column profiles rescaled to a Euclidean metric and orthogonally rotated to a principal axes orientation.

CA Example. Figure 4 contains a plot of the results of a simple CA of a survey of car owners. The questions included origin of the car (American, Japanese, European), and marital/family status (single, married, single and living with children, and married living with children). Both variables are categorical. Table 1 contains the crosstabulation and the observed minus expected frequencies. It can be seen from the observed minus expected frequencies that four cells have values appreciably different from zero (Married w Kids/American, Single/American, Married w Kids/Japanese, Single/Japanese). More people who are married with children drive American cars than would be expected if the rows and columns are independent, and more people who are single with no children drive Japanese cars than would be expected if the rows and columns are independent.

Table 1
Simple Correspondence Example Input

	Contingency Table			Observed Minus Expected Values		
	American	European	Japanese	American	European	Japanese
Married	37	14	51	-1.5133	0.4602	1.0531
Married w Kids	52	15	44	10.0885	0.2655	-10.3540
Single	33	15	63	-8.9115	0.2655	8.6460
Single w Kids	6	1	8	0.3363	-0.9912	0.6549

CA graphically shows the information in the observed minus expected frequencies. The right side of Figure 4 shows the association between being married with children and owning an American Car. The left side of the plot shows the association between being single and owning a Japanese Car. This interpretation is based on points being located in approximately the same direction from the origin and in approximately the same region of the space. Distances between row points and column points are not defined.

Multiple Correspondence Analysis. Multiple correspondence analysis (MCA) is a generalization of simple CA for more than two variables. The input is a *Burt table*, which is a partitioned symmetric matrix containing all pairs of crosstabulations among a set of categorical variables. Each diagonal partition is a diagonal matrix containing marginal frequencies (a crosstabulation of a variable with itself). Each off-diagonal partition is an ordinary contingency table. Each contingency table above the diagonal has a transposed counterpart below the diagonal. A Burt table is the inner product of a partitioned design matrix. There is one partition per categorical variable, and each partition is a binary design matrix. Each design matrix has one column per category, and a single 1 in each row. The partitioned design matrix has exactly m ones in each row, where m is the number of categorical variables. The results of a MCA of a Burt table, $\mathbf{N}$, are the same as the column results from a simple CA of the design matrix whose inner product is the Burt table. MCA is not a simple CA of the Burt table. The coordinates for MCA are $\mathbf{D_c}^{-1}\mathbf{B}\mathbf{D_u}$, from $(1/f)\mathbf{N} = \mathbf{P} = \mathbf{P}' = \mathbf{B}\mathbf{D_u}^2\mathbf{B}'$, where $\mathbf{B}'\mathbf{D_c}^{-1}\mathbf{B} = \mathbf{I}$.

MCA Example. Figure 5 contains a plot of the results of an MCA of a survey of car owners. The questions included origin of the car (American, Japanese, European), size of car (small, medium, large), type of car (family, sporty, work vehicle), home ownership (owns, rents), marital/family status (single, married, single and living with children, and married living with children), and sex (male, female). The variables are all categorical.

The top-right quadrant of the plot shows that the categories single, single with kids, 1 income, and renting a home are associated. Proceeding clockwise, the categories sporty, small, and Japanese are associated. In the bottom-left quadrant we see the association between being married, owning your own home, and having two incomes. Having children is associated with owning a large American family car. Such information could be used in marketing research to identify target audiences for advertisements. This interpretation is based on points being located in approximately the same direction from the origin

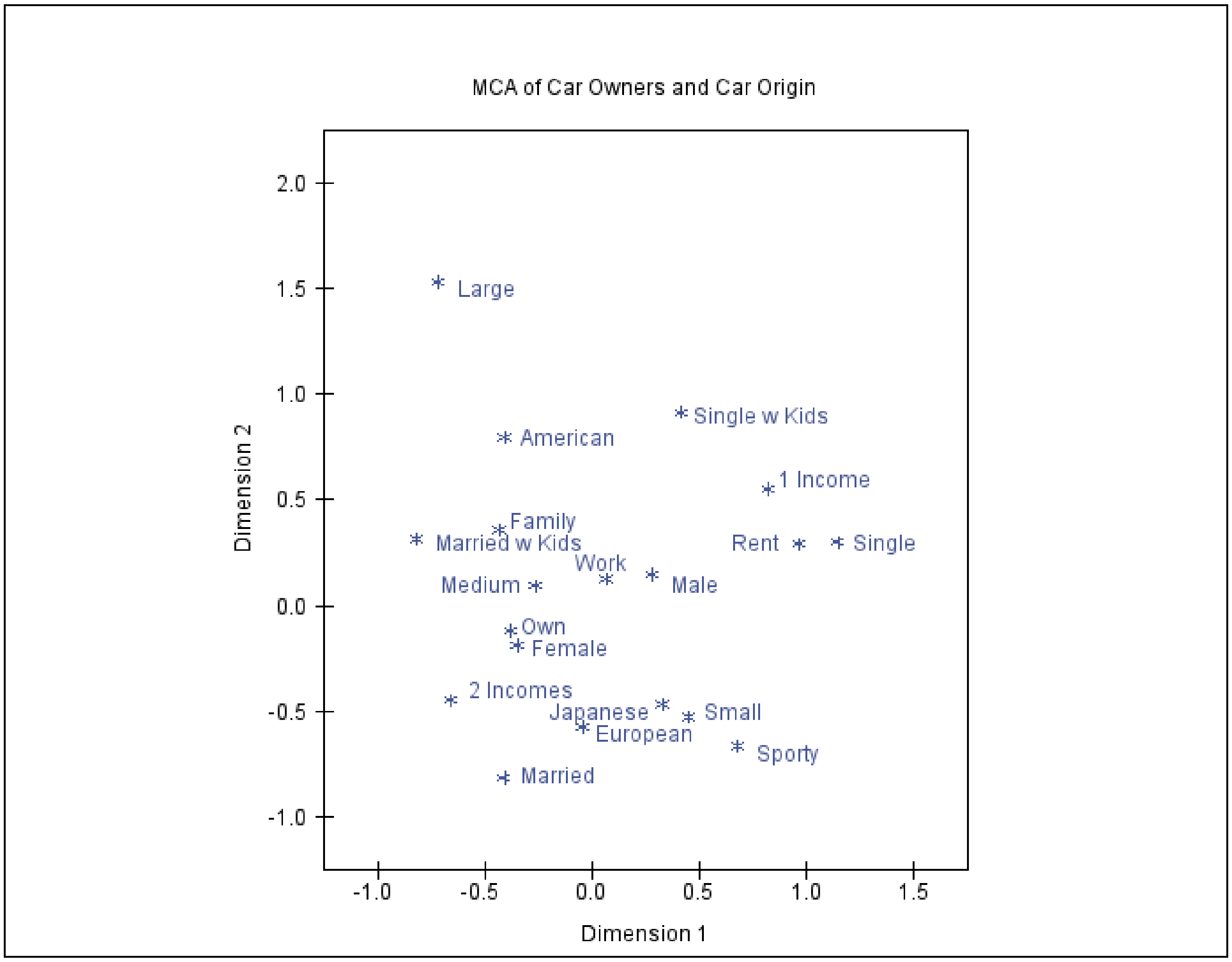


Figure 5. Multiple Correspondence Analysis

and in approximately the same region of the space. Distances between points are not interpretable in MCA.

Other CA Standardizations. Other standardizations have been proposed for CA by several authors. The usual goal is to provide a standardization that avoids the problem of row and column distances being undefined. Unfortunately, this problem remains unsolved. Carroll, Green, and Schaffer (1986) proposed that simple CA coordinates should be transformed to MCA coordinates before they are plotted. They argued that all distances are then comparable, but Greenacre (1989) showed that their assertion was incorrect. Others have also claimed to have discovered a method of defining the between row and column differences, but so far no method has been demonstrated to be correct.

Notes

The Geometry of the Scatter Plots. All of the scatter plots in this chapter were created with the axes equated so that a centimeter on the y-axis represents the same data range as a centimeter on the x-axis. *This is important.* Distances, angles between vectors, and projections are evaluated to interpret the plots. When the axes are equated, distances and angles are correctly represented in the plot. When axes are scaled independently, for example to fill the page, then the correct geometry is not presented. The important step of equating the axes is often overlooked in practice.

In a true biplot, $\mathbf{A} = \mathbf{U}\mathbf{D}^r$ and $\mathbf{B} = \mathbf{V}\mathbf{D}^{(1-r)}$ are plotted, and the elements of $\mathbf{Y}$ can be approximated from $y_{ij} \approx \mathbf{a}_i'\mathbf{b}_j$. For MDPREF and PREFMAP, the absolute lengths of the vectors are not important since the goal is to project points on vectors, not look at scalar products of row points and column vectors. It is often necessary to change the lengths of *all* of the vectors to improve the graphical display. If all of the vectors are relatively short with end points clustered near the origin, the display will not look good. To avoid this problem in Figure 1, *both* the x-axis and y-axis coordinates were multiplied by the constant 2.5, to lengthen all vectors by the same relative amount. The coordinates must not be scaled independently.

Software. All data analyses were performed with Release 8.00 of the SAS System. MDPREF is performed with PROC PRINQUAL, simple and multiple correspondence analysis are performed with PROC CORRESP, and PREFMAP is performed with PROC TRANSREG. The plots are prepared with the SAS `%PlotIt` autocall macro. If your site has installed the autocall libraries supplied by SAS Institute and uses the standard configuration of SAS software supplied by the Institute, you need only to ensure that the SAS system option `mautosource` is in effect to begin using the autocall macros. See pages 803–805 and pages 1178–1210.

Conclusions

Marketing research helps marketing decision makers understand their customers and their competition. Correspondence analysis compactly displays survey data to aid in determining what kinds of consumers are buying certain products. Multidimensional preference analysis shows product positioning, group preferences, and individual preferences. The plots may suggest how to reposition products to appeal to a broader audience. They may also suggest new groups of customers to target. Preference mapping is used as an aid in understanding MDPREF and multidimensional scaling results. PREFMAP displays product attributes in the same plot as the products. The insight gained from perceptual mapping methods can be a valuable asset in marketing decision making. These techniques can help marketers gain insight into their products, their customers, and their competition.

Concluding Remarks

I hope you like this book, the new options, the new presentation style, and the new and improved macros. In particular, I hope you find the `%MktEx` and `%ChoicEff` macros to be very powerful and useful. My goal in writing this book and tool set is to help you do better research and do it more quickly and more easily. I would like to hear what you think. Many of my examples and enhancements to the software are based on feedback from people like you. If you have comments or suggestions for future revisions, write Warren F. Kuhfeld, (Warren.Kuhfeld at sas.com) at SAS Institute Inc. My goal to provide you with enough examples so that you can easily adapt aspects of one or more examples to fit your particular needs. When I do not succeed, tell me about it and I will try to add a new example to the next revision. Please direct questions to the Technical Support Division (see page 25) and suggestions to me. Please email me. I would like to hear from you!

References

Addelman, S. (1962a), "Orthogonal Main-Effects Plans for Asymmetrical Factorial Experiments," *Technometrics*, 4, 21–46.

Addelman, S. (1962b), "Symmetrical and Asymmetrical Fractional Factorial Plans," *Technometrics*, 4, 47–58.

Agresti, A. (1990), *Categorical Data Analysis.* New York: John Wiley and Sons.

Anderson, D.A. and Wiley, J.B. (1992), "Efficient Choice Set Designs for Estimating Cross-Effects Models," *Marketing Letters*, 3, 357–370.

Anderson, D.A. (2003), personal communication.

de Boor, C. (1978), *A Practical Guide to Splines*, New York: Springer Verlag.

Bose, R.C. (1947), "Mathematical Theory of the Symmetrical Factorial Design," *Sankhya*, 8, 107–166.

Booth, K.H.V. and Cox, D.R. (1962), "Some Systematic Super-Saturated Designs," *Technometrics*, 4, 489–495.

Breiman, L. and Friedman, J.H. (1985), "Estimating Optimal Transformations for Multiple Regression and Correlation," (with discussion), *Journal of the American Statistical Association*, 77, 580–619.

Breslow, N. and Day, N.E. (1980), *Statistical Methods in Cancer Research, Vol. II: The Design and Analysis of Cohort Studies,* Lyon: IARC.

Bunch, D.S., Louviere, J.J., and Anderson, D.A. (1996), "A Comparison of Experimental Design Strategies for Choice-Based Conjoint Analysis with Generic-Attribute Multinomial Logit Models," Working Paper, Graduate School of Management, University of California at Davis.

van der Burg, E. and de Leeuw, J. (1983), "Non-linear Canonical Correlation," *British Journal of Mathematical and Statistical Psychology*, 36, 54–80.

Carmone, F.J. and Green, P.E. (1981), "Model Misspecification in Multiattribute Parameter Estimation," *Journal of Marketing Research*, 18 (February), 87–93.

Carroll, J.D. (1972), "Individual Differences and Multidimensional Scaling," in *Multidimensional Scaling: Theory and Applications in the Behavioral Sciences (Volume 1)*, in Shepard, R.N., Romney, A.K., and Nerlove, S.B. (ed.), New York: Seminar Press.

Carroll, J.D, Green, P.E., and Schaffer, C.M. (1986), "Interpoint Distance Comparisons in Correspondence Analysis," *Journal of Marketing Research*, 23, 271–280.

Carson, R.T., Louviere, J.J, Anderson, D.A., Arabie, P., Bunch, D., Hensher, D.A., Johnson, R.M., Kuhfeld, W.F., Steinberg, D., Swait, J., Timmermans, H., and Wiley, J.B. (1994), "Experimental Analysis of Choice," *Marketing Letters,* 5(4), 351–368.

Chakravarti, I.M. (1956), "Fractional Replication in Asymmetrical Factorial Designs and Partially Balanced Arrays," *Sankhya*, 17, 143–164.

Chrzan, K. and Elrod, T. (1995), "Partial Profile Choice Experiments: A Choice-Based Approach for Handling Large Numbers of Attributes," paper presented at the AMA's 1995 Advanced Research Techniques Forum, Monterey, CA.

Colbourn, C.J. and de Launey, W. (1996), "Difference Matrices," in C.J. Colbournand J.H. Dinitz, *The CRC Handbook of Combinatorial Designs*, New York, CRC Press Inc.

Cook, R.D. and Nachtsheim, C.J. (1980), "A Comparison of Algorithms for Constructing Exact *D*-optimal Designs," *Technometrics*, 22, 315–324.

Cook, R.D. and Nachtsheim, C.J. (1989), "Computer-Aided Blocking of Factorial and Response-Surface Designs," *Technometrics* 31 (August), 339–346.

Coolen, H., van Rijckevorsel, J., and de Leeuw, J. (1982), "An Algorithm for Nonlinear Principal Components with B-splines by Means of Alternating Least Squares," in H. Caussinus, P. Ettinger, and R. Tomassone (ed.), *COMPUSTAT 1982*, Part 2, Vienna: Physica Verlag.

Dawson, J. (1985), "A Construction for Generalized Hadamard Matrices, GH(4q, EA(q))," *Journal of Statistical Planning and Inference*, 11, 103–110.

De Cock, D. and Stufken, J. (2000), "On Finding Mixed Orthogonal Arrays of Strength 2 With Many 2-Level Factors," *Statistics and Probability Letters*, 50, 383–388.

de Launey, W. (1986), "A Survey of Generalized Hadamard Matrices and Difference Matrices D(k, λ; G) with Large k," *Utilitas Mathematica*, 30, 5–29.

de Launey, W. (1987), *(O,G)-Designs and Applications*, Dissertation, University of Sydney.

de Launey, W. (1987), "On Difference Matrices, Transversal Designs, Resolvable Traversal Designs and Large Sets of Mutually Orthogonal F-Squares," *Journal of Statistical Planning and Inference*, 16, 107–125.

Dey, A. (1985), *Orthogonal Fractional Factorial Designs*, New York: Wiley.

DuMouchell, W. and Jones, B. (1994), "A Simple Bayesian Modification of *D*-Optimal Designs to Reduce Dependence on an Assumed Model," *Technometrics* 36 (February), 37–47.

Dykstra, O. (1971), "The Augmentation of Experimental Data to Maximize $|(\mathbf{X}'\mathbf{X})^{-1}|$," *Technometrics*, 13 (August), 682–688.

Eckart, C. and Young, G. (1936), "The Approximation of One Matrix by Another of Lower Rank," *Psychometrika*, 1, 211–218.

Elliott, J.E.H. and Butson, A.H. (1966), "Relative Difference Sets," *Illinois J. Math.*, 10, 517–531.

Elrod, T., Louviere, J.J, and Davey, K.S. (1992), "An Empirical Comparison of Ratings-Based and Choice-Based Conjoint Models," *Journal of Marketing Research*, 29 (August), 368–377.

Ehrlich, H. (1964), "Determinantenabschatzungen fur Binare Matrizen," *Math. Z.* 83, 123–132.

Fedorov, V.V. (1972), *Theory of Optimal Experiments*, translated and edited by W.J. Studden and E.M. Klimko, New York: Academic Press.

Finkbeiner, C.T. (1988), "Comparison of Conjoint Choice Simulators," Sawtooth Software Conference Proceedings.

Finn, A. and Louviere, J.J. (1992), "Determining the Appropriate Response to Evidence of Public Concern: The Case of Food Safety," *Journal of Public Policy and Marketing*, 11, 1, 12–25.

Fisher, R. (1938), *Statistical Methods for Research Workers*, 10th Edition, Edinburgh: Oliver and Boyd Press.

Gabriel, K.R. (1981), “Biplot Display of Multivariate Matrices for Inspection of Data and Diagnosis,” *Interpreting Multivariate Data*, V. Barnett (ed.), London: John Wiley and Sons, Inc.

Gail, M.H., Lubin, J.H., and Rubinstein, L.V. (1981), “Likelihood Calculations for Matched Case-control Studies and Survival Studies with Tied Death Times,” *Biometrika*, 68, 703–707.

Gifi, A. (1981), *Nonlinear Multivariate Analysis*, Department of Data Theory, The University of Leiden, The Netherlands.

Gifi, A. (1990), *Nonlinear Multivariate Analysis*, New York: Wiley.

Green, P.E. (1974), “On the Design of Choice Experiments involving Multifactor Alternatives,” *Journal of Consumer Research*, 1, 61–68.

Green, P.E. and Rao, V.R. (1971), “Conjoint Measurement for Quantifying Judgmental Data,” *Journal of Marketing Research*, 8, 355–363.

Green, P.E. and Srinivasan, V. (1990), “Conjoint Analysis in Marketing: New Developments with Implications for Research and Practice,” *Journal of Marketing*, 54, 3–19.

Green, P.E. and Wind, Y. (1975), “New Way to Measure Consumers’ Judgments,” *Harvard Business Review*, July–August, 107–117.

Green, P.E. and Rao, V.R. (1971), “Conjoint Measurement for Quantifying Judgmental Data,” *Journal of Marketing Research*, 8, 355–363.

Greenacre, M.J. (1984), *Theory and Applications of Correspondence Analysis*, London: Academic Press.

Greenacre, M.J. (1989), “The Carroll-Green-Schaffer Scaling in Correspondence Analysis: A Theoretical and Empirical Appraisal,” *Journal of Market Research*, 26, 358–365.

Greenacre, M.J. and Hastie, T. (1987), “The Geometric Interpretation of Correspondence Analysis,” *Journal of the American Statistical Association*, 82, 437–447.

Hadamard, J. (1893), “Resolution d’une Question Relative aux Determinants,” *Bull. des Sciences Math,* (2), 17, 240–246.

Hastie, T. and Tibshirani, R. (1986), “Generalized Additive Models,” *Statistical Science*, 3, 297–318.

Hedayat, A.S., Sloane, N.J.A., and Stufken, J. (1999), *Orthogonal Arrays*, New York: Springer.

Hedayat, A.S., and Wallis, W.D. (1978), “Hadamard Matrices and Their Applications,” *Ann. Stat.*, 6, 1184–1238.

Hoffman, D.L., and Franke, G.R. (1986), “Correspondence Analysis: Graphical Representation of Categorical Data in Marketing Research,” *Journal of Marketing Research*, 23, 213–227.

Hoffman, S.D. and Duncan, G.J. (1988), “Multinomial and Conditional Logit Discrete-choice Models in Demography,” *Demography,* 25 (3), 415–427.

Huber, J. and Zwerina, K. (1996), “The Importance of Utility Balance in Efficient Choice Designs,” *Journal of Marketing Research*, 33, 307–317.

Huber, J., Wittink, D.R., Fiedler, J.A., and Miller, R. (1993), "The Effectiveness of Alternative Preference Elicitation Procedures in Predicting Choice," *Journal of Marketing Research*, 30 (February), 105–114.

Kharaghania, H., and Tayfeh-Rezaiea, B. (2004), "A Hadamard Matrix of Order 428," [`http://math.ipm.ac.ir/tayfeh-r/papersandpreprints/h428.pdf`].

Kirkpatrick, S., Gellat, C.D., and Vecchi, M.P. (1983), "Optimization by Simulated Annealing," *Science*, 220, 671–680.

Krieger, A.B. and Green, P.E. (1991), "Designing Pareto Optimal Stimuli for Multiattribute Choice Experiments," *Marketing Letters*, 2, 337–348.

Kruskal, J.B. and Wish, M. (1978), *Multidimensional Scaling*, Sage University Paper series on Quantitative Applications in the Social Sciences, 07–011, Beverly Hills and London: Sage Publications.

Kruskal, J.B. and Shepard, R.N. (1974), "A Nonmetric Variety of Linear Factor Analysis," *Psychometrika*, 38, 123–157.

Kuhfeld, W.F. (1990), *SAS Technical Report R-108: Algorithms for the PRINQUAL and TRANSREG Procedures*, [`http://support.sas.com/publishing/pubcat/techreports/59040.pdf`], Cary NC: SAS Institute Inc.

Kuhfeld, W.F. (1991), "A Heuristic Procedure for Label Placement in Scatter Plots," Presented to the joint meeting of the Psychometric Society and Classification Society of North America, Rutgers University, New Brunswick NJ, June 13–16, 1991.

Kuhfeld, W.F. (2010), "Marketing Research Methods in SAS," [`http://support.sas.com/resources/papers/tnote/tnote_marketresearch.html`].

Kuhfeld, W.F. (2005), "Difference Schemes via Computerized Searches," *Journal of Statistical Planning and Inference*, 127, 1–2, 341–346.

Kuhfeld, W.F. and Garratt, M. (1992), "Linear Models and Conjoint Analysis with Nonlinear Spline Transformations," Paper presented to the American Marketing Association Advanced Research Techniques Forum, Lake Tahoe, Nevada.

Kuhfeld, W.F. and Suen, C.Y. (2005), "Some new orthogonal arrays OA($4r$, $r^1 2^p$, 2)," *Statistics and Probability Letters*, 75, 169–178.

Kuhfeld, W.F. and Tobias, R.D. (2005), "Large Factorial Designs for Product Engineering and Marketing Research Applications," *Technometrics*, 47, 132–141.

Kuhfeld, W.F., Tobias, R.D., and Garratt, M. (1994), "Efficient Experimental Design with Marketing Research Applications," *Journal of Marketing Research*, 31, 545–557.

Lazari, A.G. and Anderson, D.A. (1994), "Designs of Discrete Choice Set Experiments for Estimating Both Attribute and Availability Cross-Effects," *Journal of Marketing Research*, 31, 375–383.

Lebart, L., Morineau, A., and Warwick, K.M. (1984), *Multivariate Descriptive Statistical Analysis: Correspondence Analysis and Related Techniques for Large Matrices*, New York: Wiley.

de Leeuw, J. (1986), "Regression with Optimal Scaling of the Dependent Variable," Department of Data Theory, The University of Leiden, The Netherlands.

de Leeuw, J., Young, F.W., and Takane, Y. (1976), "Additive Structure in Qualitative Data: an Alternating Least Squares Method with Optimal Scaling features," *Psychometrika*, 41, 471–503.

Louviere, J.J. (1988), *Analyzing Decision Making, Metric Conjoint Analysis*, Sage University Papers, Beverly Hills: Sage.

Louviere, J.J. (1991), "Consumer Choice Models and the Design and Analysis of Choice Experiments," Tutorial presented to the American Marketing Association Advanced Research Techniques Forum, Beaver Creek, Colorado.

Louviere, J.J. (1991), "Best-Worst Scaling: A Model for the Largest Difference Judgments," Working Paper, University of Alberta.

Louviere, J.J. and Woodworth, G. (1983), "Design and Analysis of Simulated Consumer Choice of Allocation Experiments: A Method Based on Aggregate Data," *Journal of Marketing Research*, 20 (November), 350–367.

Manski, C.F. and McFadden, D. (1981), *Structural Analysis of Discrete Data with Econometric Applications.* Cambridge: MIT Press.

Mardia, K.V., Kent, J.T., and Bibby, J.M. (1979), Multivariate Analysis, New York: Academic Press.

McFadden, D. (1974), "Conditional logit Analysis Of Qualitative Choice Behavior," in P. Zarembka (ed.) *Frontiers in Econometrics,* New York: Academic Press, 105–142.

McKelvey, R.D. and Zavoina, W. (1975), "A Statistical Model for the Analysis Of Ordinal Level Dependent Variables," *Journal of Mathematical Sociology,* 4, 103–120.

Meyer, R.K. and Nachtsheim, C.J. (1995), "The Coordinate-Exchange Algorithm for Constructing Exact Optimal Experimental Designs," *Technometrics*, 37, 60–69.

Mitchell, T.J. and Miller, F.L. Jr. (1970), "Use of Design Repair to Construct Designs for Special Linear Models," *Math. Div. Ann. Progr. Rept. (ORNL-4661)*, 130–131, Oak Ridge, TN: Oak Ridge National Laboratory.

Mitchell, T.J. (1974), "An Algorithm for the Construction of D-optimal Experimental Designs," *Technometrics*, 16 (May), 203–210.

Nishisato, S. (1980), *Analysis of Categorical Data: Dual Scaling and Its Applications*, Toronto: University of Toronto Press.

Nguyen, M.V.M. (2005), *Computer-Algebraic Methods for the Construction of Designs of Experiments*, Dissertation, Eindhoven University of Technology.

Nguyen, M.V.M. (2006), An Online Service for Computing Hadamard Matrices and Orthogonal Arrays of Strength 3, [`http://www.mathdox.com/nguyen/index.jsp`].

Nguyen, N.K. (2006), OA($2^4 10^4$, 100), March 12, 2006; OA($2^{14} 6^1 14^1$, 84), March 12, 2006; OA($2^{15} 6^1 10^1$, 60), March 26, 2006; OA($2^{13} 3^2 6^1$, 36), March 28, 2006; Computerized Augmentation of Existing Arrays, (Personal Communication).

Paley, R.E.A.C. (1933), "On Orthogonal Matrices," *J. Math. Phys*, 12, 311–320.

Pang, S.Q., Zhang, Y.S., and Liu, S.Y. (2004a), "Further Results on Orthogonal Arrays Obtained by the Generalized Hadamard Product," *Statistics and Probability Letters*, 68(1), 17–25.

Pang, S.Q., and Zhang, Y.S. (2004b), “Orthogonal Arrays of Size 108 with Six-Level Columns,” *SUT Journal of Mathematics*, 40(1), 1–12.

Perreault, W.D. and Young, F.W. (1980), “Alternating Least Squares Optimal Scaling: Analysis of Nonmetric Data in Marketing Research,” *Journal of Marketing Research*, 17, 1–13.

Raktoe, B.L., Hedayat, A.S., and Federer, W.T. (1981), *Factorial Designs*, New York: John Wiley and Sons.

Ramsay, J.O. (1988), “Monotone Regression Splines in Action,” *Statistical Science*, 3, 425–461.

Rao, C.R. (1947), “Factorial Experiments Derivable from Combinatorial Arrangements of Arrays,” *Journal of the Royal Statistical Society*, Suppl., 9, 128–139.

van Rijckevorsel, J. (1982), “Canonical Analysis with B-splines,” in H. Caussinus, P. Ettinger, and R. Tomassone (ed.), *COMPUSTAT 1982*, Part I, Vienna: Physica Verlag.

Sawtooth Software (2005), *The MaxDiff/Web System Technical Paper* [`www.sawtoothsoftware.com`].

Sawtooth Software (2007), *The MaxDiff/Web v6.0 Technical Paper* [`www.sawtoothsoftware.com`].

Seberry, J. and Yamada, M. (1992), “Hadamard Matrices, Sequences, and Block Designs,” in J.H. Dinitzand D.R. Stinson(ed.) *Contemporary Design Theory, A Collection of Surveys*, New York: Wiley.

Schiffman, S.S., Reynolds, M.L., and Young, F.W. (1981), *Introduction to Multidimensional Scaling*, New York: Academic Press.

Sloane, N.J.A. (2004), “A Library of Orthogonal Arrays,” [`http://www.research.att.com/~njas/oadir`].

Smith, P.L. (1979), “Splines as a Useful and Convenient Statistical Tool,” *The American Statistician*, 33, 57–62.

Spence, E. (1975), “Hadamard Matrices from Relative Difference Sets,” *J. Combinatorial Theory Ser. A*, 19, 287–300.

Spence, E. (1975), “Skew-Hadamard Matrices of the Goethals-Seidel Type,” *Canad. J. Math.*, 27, 555–560.

Spence, E. (1977), “A Family of Difference Sets,” *Combinatorial Theory Ser. A*, 22 103–106.

Spence, E. (1977), “Skew-Hadamard Matrices of Order 2(q+1),” *Discrete Mathematics*, 18, 79–85.

Steckel, J.H., DeSarbo, W.S., and Mahajan, V. (1991), “On the Creation of Acceptable Conjoint Analysis Experimental Designs,” *Decision Sciences*, 22, 435–442.

Suen, C.Y. (1989a), “A Class of Orthogonal Main Effects Plans,” *Journal of Statistical Planning and Inference*, 21, 391–394.

Street, D.J., and Burgess, L. (2007) *The Construction of Optimal Stated Choice Experiments*, New York: Wiley.

Suen, C.Y. (1989b), “Some Resolvable Orthogonal Arrays with Two Symbols,” *Communications in Statistics, Theory and Methods*, 18, 3875–3881.

Suen, C.Y. (2003a), "Construction of Mixed Orthogonal Arrays by Juxtaposition," paper in review.

Suen, C.Y. (2003b), "Table of Orthogonal Arrays," personal communication.

Suen, C.Y. (2003c), "Some Mixed Orthogonal Arrays Obtained by Orthogonal Projection Matrices," to be submitted.

Suen, C.Y. and Kuhfeld, W.F. (2005), "On the Construction of Mixed Orthogonal Arrays of Strength Two," *Journal of Statistical Planning and Inference*, 133, 555–560.

Taguchi, G. (1987), *System of Experimental Design: Engineering Methods to Optimize Quality and Minimize Costs.* White Plains, NY: UNIPIB, and Dearborn, MI: American Supplier Institute.

Tenenhaus, M. and Young, F.W. (1985), "An Analysis and Synthesis of Multiple Correspondence Analysis, Optimal Scaling, Dual Scaling, Homogeneity Analysis, and Other Methods of Quantifying Categorical Multivariate Data," *Psychometrika*, 50, 91–119.

Turyn, R.J. (1972), "An Infinite Class of Williamson Matrices," *J. Combin. Th. Ser.* A 12, 319–321.

Turyn, R.J. (1974), "Hadamard Matrices, Baumert-Hall Units, Four-Symbol Sequences, Pulse Compression, and Surface Wave Encodings," *J. Combin. Th. Ser.* A 16, 313–333.

Wang, J.C. (1996a), "Mixed Difference Matrices and the Construction of Orthogonal Arrays," *Statist. Probab. Lett.*, 28, 121–126.

Wang, J.C. (1996b), *A Recursive Construction of Orthogonal Arrays*, preprint.

Wang, J.C. and Wu, C.F.J. (1989), "An Approach to the Construction of Asymmetrical Orthogonal Arrays," *IIQP Research Report RR-89-01*, University of Waterloo.

Wang, J.C. and Wu, C.F.J. (1991), "An Approach to the Construction of Asymmetrical Orthogonal Arrays," *Journal of the American Statistical Association*, 86, 450–456.

Williamson, J. (1944), "Hadamard's Determinant Theorem and the Sum of Four Squares," Duke Math. J., 11, 65–81.

Winsberg, S. and Ramsay, J.O. (1980), "Monotonic Transformations to Additivity Using Splines," *Biometrika*, 67, 669–674.

Wittink, D.R. and Cattin, P. (1989), "Commercial Use of Conjoint Analysis: An Update," *Journal of Marketing*, 53 (July), 91–96.

Wittink, D.R., Krishnamurthi, L., and Reibstein, D.J. (1989), "The Effect of Differences in the Number of Attribute Levels in Conjoint Results," *Marketing Letters*, *1:2*, 113–123.

Xu, H. (2002), "An Algorithm for Constructing Orthogonal and Nearly Orthogonal Arrays with Mixed Levels and Small Runs," *Technometrics*, 44, 356–368.

Yamada, M. (1986), "Hadamard Matrices Generated by an Adaptation of Generalized Quaternion Type Array," *Graphs Combina.*, 2, 179–187.

Yamada, M. (1989), "Some New Series of Hadamard Matrices," *Journal of the Australian Mathematical Society*, 46, 371–383.

Young, F.W. (1981), "Quantitative Analysis of Qualitative Data," *Psychometrika*, 46, 357–388.

Young, F.W. (1987), *Multidimensional Scaling: History, Theory, and Applications*, R.M. Hamer (ed.), Hillsdale, NJ: Lawrence Erlbaum Associates.

Young, F.W., de Leeuw, J., and Takane, Y. (1976), "Regression with Qualitative and Quantitative Variables: an Alternating Least Squares Approach with Optimal Scaling Features," *Psychometrika*, 41, 505–529.

Zhang, Y.S., Lu, Y., and Pang, S. (1999), "Orthogonal Arrays Obtained by Orthogonal Decompositions of Projection Matrices," *Statistica Sinica*, 9, 595–604.

Zhang, Y.S., Pang, S., and Wang, Y. (2001), "Orthogonal Arrays Obtained by Generalized Hadamard Product," *Discrete Mathematics*, 238, 151–170.

Zhang, Y.S., Duan, L., Lu. Y., Zheng, Z. (2002), "Construction of Generalized Hadamard Matrices," *Journal of Statistical Planning and Inference*, 104, 239–258.

Zhang, Y.S., Weiguo, L., Meixia, M., and Zheng, Z. (2005), "Orthogonal Arrays Obtained by Generalized Kronecker Product," in review, *Journal of Statistical Planning and Inference.*

Zhang, Y.S. (2004–2006), Difference Schemes: D(24, 20, 4), March 24, 2004; D(42, 18, 7), December 28, 2005; D(48, 10, 6), January 19, 2006; D(150, 150, 3), January 21, 2006; (Personal Communication).

Zhang, Y.S. (2007), "Orthogonal Arrays Obtained By Repeating-Column Difference Matrices," *Discrete Mathematics*, 307, 246–261.

Index