R (and S-PLUS) Manual to Accompany Agresti's *Categorical Data Analysis* (2002) 2nd edition

Laura A. Thompson, 2009©

Table of Contents

Introduction and Changes from First Edition

This manual accompanies Agresti's *Categorical Data Analysis* (2002). It provides assistance in doing the statistical methods illustrated there, using S-PLUS and the R language. Although I have used the Windows versions of these two softwares, I suspect there are few changes in order to use the code in other ports. I have included examples of almost all of the major (and some minor) analyses introduced by Agresti. The manual chapters parallel those from Agresti so that, for example, in Chapter 2 I discuss using the software to conduct analyses from Agresti's Chapter 2. In most cases I use the data provided in the text. There are only one or two occasions where I use data from the problems sections in Agresti. Discussion of results of analyses is brief since the discussion appears in the text. That said, one of the improvements in the current manual over the previous version of the manual (Thompson, 1999) is that it is more self-contained. In addition, I include a summary of the corresponding chapter from Agresti at the beginning of each chapter in the manual. However, it will still be helpful to refer to the text to understand completely the analyses in this manual.

In the manual, I frequently employ functions that come from user-contributed libraries (packages) of S-PLUS (or R). In the text, I note when a particular library or package is used. These libraries are not automatically included with the software, but can be easily downloaded from the internet, especially for the newest version of R for Windows. I mention in the next section how to get these libraries and how to install them for use. Many times, the library or package has its own help manual or help section. I will demonstrate how to access these from inside the software.

I used S-PLUS 6.1 through 7.0 for Windows and R versions 1.8 through 2.8.1 for Windows for the analyses. However, S-PLUS for Windows versions as far back as 3.0 will do many of the analyses (but not all). This is not so easily said for R, as user-contributed packages frequently apply to the newer versions of R (e.g., at least 1.3.0). Many of the analyses can be applied to either S-PLUS or R. Some need small changes in order to apply to both softwares; these changes I have provided where necessary. In general, when I refer to both of the softwares, I will use the "generic" name, S. *Also, if I do not indicate that I am using either S-PLUS or R for a particular command, that means it will work in both softwares.*

To separate input to R or S-PLUS and output from other text in this manual, I have put normal text in Arial font and commands and output in `courier` font. The input commands are in **`bold`** font, whereas the output is not. Also, all assignments will use the "`<-`" convention instead of "`=`" (or, "_").

Finally, this manual assumes some familiarity in using the basic commands of S. To keep the manual from being too long I do not discuss at great length functions that I use which are not directly related to categorical data analysis. See Section H below for information on obtaining introductory documentation for R or S-PLUS.

A. Obtaining the R Software for Windows

The language (and associated software interface) R can loosely be described as "open-source" S. It is downloadable from the site http://cran.r-project.org. Information on how to install R, as well as several PDF documents and user-contributed documents on the language and its features are included on the website.

B. Libraries in S-PLUS and Packages in R

The S-PLUS libraries used in this manual that do not come with the software are

MASS (B. Ripley) - (used throughout)
Multiv (F. Murtagh) - (used for correspondence analysis)
cond (A Brazzale) – (used for conditional logistic regression in chapter 6, NOTE: no longer supported) http://www.ladseb.pd.cnr.it/~brazzale/lib.html#ins
Design (F. Harrell) - (used throughout)
Hmisc (F. Harrell) - (support for Design library)
nnet (B. Ripley) - for the function `multinom` (chapter 7, multinomial logit models)
nolr (M. Mathieson) - (nonlinear ordinal models - supplement to chapter 9)
rmtools (A. Azzalini & M. Chiogna) - (used for chapter 11)

yags2 (V. Carey) - (used for chapter 11)

Most of these libraries can be obtained in .zip form from URL http://lib.stat.cmu.edu/DOS/S/Swin or http://lib.stat.cmu.edu/DOS/S. Currently, the URL http://www.stats.ox.ac.uk/pub/MASS4/Winlibs/ contains many ports to S-PLUS 6.0 for Windows. To install a library, first download it to any folder on your computer. Next, "unzip" the file using an "unzipping" program. This will extract all the files into a new folder in the directory into which you downloaded the zip file. Move the entire folder to the library directory under your S-PLUS directory (e.g., c:/program files/Insightful/splus61/library).

To load a library, you can either pull down the File menu in S-PLUS and select Load Library or type one of the following in a script or command window

```
library("libraryname",first=T)    # loads libraryname into first database position
library(libraryname)
```

To use the library's help manual from inside S-PLUS or R type in a script or command window

```
help(library="libraryname")
```

Many of the R packages used in this manual that do not come with the software are listed below (not a compete list)

MASS – (VR bundle, Venables and Ripley)
rmutil (J. Lindsey) – (used with gnlm) http://alpha.luc.ac.be/~lucp0753/rcode.html
gnlm (J. Lindsey) – http://alpha.luc.ac.be/~lucp0753/rcode.html
repeated (J. Lindsey) – http://alpha.luc.ac.be/~lucp0753/rcode.html
SuppDists (B. Wheeler) – (used in chapter 1)
combinant (V. Carey) – (used in chapters 1, 3)
methods – (used in chapter 3)
Bhat (E. Luebeck)– (used throughout)
mgcv (S. Wood) – (used for fitting GAM models)
modreg (B. Ripley) – (used for fitting GAM models)
gee and geepack (J. Yan) – (used in chapter 11)
yags (V. Carey) – (used in chapter 11)
gllm – (used for generalized log linear models and latent class models)
GlmmGibbs (Myles and Clayton) – (used for generalized linear mixed models, chap. 12)
glmmML (G. Broström) – (used for generalized linear mixed models, chapter 12)
CoCoAn (S. Dray) – (used for correspondence analysis)
e1071 (A. Weingessel) – (used for latent class analysis)
vcd (M. Friendly)– (used in chapters 2, 3, 5, 9 and 10)
brlr (D. Firth) – (used in chapter 6)
BradleyTerry (D. Firth) – (used in chapter 10)
ordinal (P. Lindsey) – (used in chapter 7) http://popgen.unimaas.nl/~plindsey/rlibs.html
design (F. Harrell) – (used throughout) http://hesweb1.med.virginia.edu/biostat
Hmisc (F. Harrell) – (used throughout)
VGAM (T. Yee) – (used in chapters 7 and 9)
http://www.stat.auckland.ac.nz/~yee/VGAM/download.shtml
mph (J. Lang) – (used in chapters 7, 10)
http://www.stat.uiowa.edu/~jblang/mph.fitting/mph.fit.documentation.htm#availability
exactLoglinTest (B. Caffo) – (used in chapters 9, 10)
http://www.biostat.jhsph.edu/~bcaffo/downloads.htm
aod – used in chapter 13
lca – used in chapter 13
mmlcr – used in chapter 13
flexmix – used in chapter 13
npmlreg – used in chapter 13

Rcapture – used in chapter 13

R packages can be installed from Windows using the `install.packages` function. This function can be called from the console or from the pull-down "Packages" menu. A package is loaded in the same way as for S-PLUS. As well, the command `help(package=pkg)` can used to invoke the help menu for package `pkg`.

To detach a library or package, named `library.name` or `pkg`, respectively, one can issue the following commands, among others.

```
detach("library.name")      # S-PLUS
detach("package:pkg")       # R
```

C. Setting contrast types using `Options()`

The `options` function can be used to set the type of contrasts used in estimating models. The default for S-PLUS is Helmert contrasts for factors and polynomial contrasts for ordered factors. The default for R is treatment contrasts for factors and polynomial contrasts for ordered factors. I use treatment contrasts for factors for most of the analysis so that they match Agresti's estimates. However, there are times when I use sum-to-zero contrasts (`contr.sum`). The type of contrasts I use is usually indicated by a call to `options` prior to fitting the model, if necessary. If the call to `options` has been omitted, please assume I am using treatment contrasts for factors.

One can find out exactly what the contrasts are in a `glm`-type fit by using the functions `model.matrix` and `contrasts`. Issuing the comand `contrasts(model.matrix(fit))` gives the contrasts.

D. Credit for functions

The author of a function is named if the function was not written by me. Whenever I use functions that do not come with S, I will mention the S-PLUS library or R package that includes them. I also give a URL, if applicable.

E. Editing functions

In several places in the text, I mention creating functions or editing existing functions that come with either S-PLUS or R. There are many ways to edit or create a function. Some of them are specific to your platform. However, one procedure that works on all platforms from the command line is a call to `fix`. For example, to edit the method function `update.default`, and call the new version `update.crosstabs`, type the following at the command line (R or S-PLUS)

```
update.crosstabs<-fix(update.default)
```

This will bring up the function code for `update.default` in a text editor from which you can make changes to the function, save them, and exit the editor. The changes will be incorporated in `update.crosstabs`. Note that the function `edit` works in mostly the same way here, but is actually a generic function that allows editing of not just function objects, but all other S objects as well.

To create a function from scratch, put the name of the new function as the argument to `fix`. For example,

```
fix(my.new.function)
```

To create functions from a script file (e.g., S-PLUS) or another editing program, one general procedure is to source the script file using e.g.,

```
source("c:/path/name.of.script.file")
```

F. A note about using S-PLUS Menus

Many of the more common methods I will illustrate can be accomplished via the S-PLUS menus. If you want to know what code corresponds to a particular menu command, issue the menu command and call up the History window (using the Window menu). All commands you have issued from menus will be there in (`gui`) code form which can then be used in a command window or script.

G. Notice of errors

All code has been tested, but there are undoubtedly still errors. Please notify me of any errors in the manual or of easier ways to perform tasks. My email address is lthompson10@yahoo.com.

H. Introductions to the S Language

This manual assumes some working knowledge of the S language. There is not space to also describe it. Fortunately, there are many tutorials available for learning S. Some of them are listed in your User's Guides that come with S-PLUS. Others are listed on the CRAN website for R (see Section A above).

I. References

Agresti, A. (2002). *Categorical Data Analysis 2nd edition*. Wiley.

Bishop, C. (1995). *Neural Networks for Pattern Recognition*. Cambridge University Press

Chambers, J. (1998). *Programming with Data*. Springer-Verlag.

Chambers, J. and Hastie, T. (1992). *Statistical Models in S*. Chapman & Hall.

Ewens, W. and Grant. G. (2001) *Statistical Methods in Bioinformatics*. Springer-Verlag.

Gentleman, R. and Ilhaka, R. (2000). "Lexical Scope and Statistical Computing." *Journal of Computational and Graphical Statistics*, 9, 491-508.

Green, P. and Silverman, B. (1994). *Nonparametric Regression and Generalized Linear Models*, Chapman & Hall.

Fletcher, R. (1987). *Practical Methods of Optimization*. Wiley.

Harrell, F. (1998). *Predicting Outcomes: Applied Survival Analysis and Logistic Regression*. Unpublished manuscript. Now available as *Regression Modeling Strategies : With Applications to Linear Models, Logistic Regression, and Survival Analysis*. Springer (2001).

Liao, L. and Rosen, O. (2001). "Fast and Stable Algorithms for Computing and Sampling From the Noncentral Hypergeometric Distribution." *The American Statistician*, 55, 366-369.

Lloyd, C. (1999). *Statistical Analysis of Categorical Data*, Wiley.

McCullagh, P. and Nelder, J. (1989). *Generalized Linear Models*, 2nd ed., Chapman & Hall.

Ripley B. (2002). On-line complements to *Modern Applied Statistics with SPLUS* (http://www.stats.ox.ac.uk/pub/MASS4).

Ross, S. (1997). *Introduction to Probability Models*. Addison-Wesley.

Selvin, S. (1998). *Modern Applied Biostatistical Methods Using SPLUS*. Oxford University Press.

Sprent, P. (1997). *Applied Nonparametric Statistical Methods*. Chapman & Hall.

Thompson, L. (1999) *S-PLUS Manual to Accompany Agresti's (1990) Catagorical Data Analysis*. (http://math.cl.uh.edu/~thompsonla/5537/Splusdiscrete.PDF).

Venables, W. and Ripley, B. (2000). *S programming*. Springer-Verlag.

Venables, W. and Ripley, B. (2002). *Modern Applied Statistics with S*. Springer-Verlag.

Wickens, T. (1989). *Multiway Contingency Tables Analysis for the Social Sciences*. LEA.

J. Acknowledgements

Thanks to Gregory Rodd and Frederico Zanqueta Poleto for reviewing the manuscript and finding many errors that I overlooked. Remaining errors are the sole responsibility of the author.

Chapter 1: Distributions and Inference for Categorical Data

A. Summary of Chapter 1, Agresti

In Chapter 1, Agresti introduces categorical data, including its types, distributions and statistical inference. Categorical variables measured on a nominal scale take values that do not have a natural ordering, whereas categorical variables measured on an ordinal scale take values that do have an ordering, but not necessarily a numerical ordering. Categorical variables are sometimes called discrete variables because they only take on a discrete or countable number of values.

Agresti discusses three main distributions for categorical variables: *binomial*, *multinomial*, and *Poisson*. The binomial distribution describes the distribution of the sum of a fixed number of independent Bernoulli trials (i.e., binary outcomes), where the probability of success is fixed across trials. The multinomial distribution extends the binomial distribution to handle trials with possibly more than two outcomes. The Poisson distribution describes the distribution of the number of events occurring in a specified length of time or space, where the intervals of time or space are independent with respect to the occurrence of an event. Formal assumptions related to the Poisson distribution (which derives from the homogeneous Poisson process) can be found in any text on stochastic processes (see, for example, Ross, 1997). A nice explanation, however, can also be found in Ewens and Grant (2001).

When observations modeled as coming from a binomial or Poisson distribution are much more variable than that predicted by the respective theoretical distributions, the use of these distributions can lead to poor predictions because the excess variability is not considered. The prescence of this excess variability is called *overdispersion*. There are options for dealing with overdispersion when it is suspected in a data set that otherwise could be reasonably modeled using a conventional distribution like the binomial or Poisson. One option is to incorporate random effects into the model (see Agresti, Chapter 12). Another option is to use a distribution with a greater variance than that dictated by the binomial or Poisson (e.g., the beta-binomial distribution is a mixture of binomials with different probabilities of success).

Given a particular probability distribution to describe the data, the likelihood function is the probability of the data as a function of the parameters. The maximum likelihood estimate (MLE) is the value of the parameter that maximizes this function. Thus, the MLE is the value for the set of parameters that give the observed data the highest probability of occurrence. Inference for categorical data deals with the MLE and tests derived from maximum likelihood. Tests of the null hypothesis of zero-valued parameters can be carried out via the Wald Test, Likelihood Ratio Test, and the Score Test. These tests are based on different aspects of the likelihood function, but are asymptotically equivalent. However, in smaller samples they yield different answers and are not equally reliable.

Agresti also discusses confidence intervals derived from "inverting" the above three tests. As mentioned on p. 13 of Agresti, "a 95% confidence interval for β is the set of β_0 for which the test of H_0: $\beta = \beta_0$ has a *P*-value exceeding 0.05." That is, it describes the set of β_0 values for which we would "keep" the null hypothesis (assuming a significance level of 0.05).

The chapter concludes with illustration of statistical inference for binomial and multinomial parameters.

B. Discrete Probability Distributions in S-PLUS and R

S-PLUS and R come with support for many built-in probability distributions, and support for many specialized distributions can be downloaded from statlib or from the CRAN website (see Introduction Section). Each distribution has functions for obtaining cumulative probabilities, density values, quantiles, and realizations of random variates. For example, in both S-PLUS and R

`dbinom(x, size, prob)` computes the density of the indicated binomial distribution at `x`
`pbinom(x, size, prob)` computes the cumulative density of the indicated binomial distribution

at `x`

`qbinom(p, size, prob)` computes the `p`th quantile of the indicated binomial distribution

`rbinom(n, size, prob)` draws `n` random variates from the indicated binomial distribution

Some of the other discrete distributions included with S-PLUS and R are the Poisson, negative binomial, geometric, hypergeometric, and discrete uniform. `rnegbin` is included with the MASS library and can generate negative binomial variates from a distribution with nonintegral "size". The beta-binomial is included in R with packages `rmutil` and `gnlm` (for beta-binomial regression via `gnlr`) from Jim Lindsey, as well as the R package `SuppDists` from Bob Wheeler. Both the `rmutil` package and `SuppDists` package contain functions for many other distributions as well (e.g., negative hypergeometric, generalized hypergeometric, beta-negative binomial, and beta-pascal or beta-geometric). The library `wle` in R has a function for generating from a discrete uniform using `runif`. The library `combinat` in R gives functions for the density of a multinomial random vector (`dmnom`) and for generating multinomial random vectors (`rmultinomial`). The `rmultinomial` function is given below

```
rmultinomial<-function (n, p, rows = max(c(length(n), nrow(p))))
{
    rmultinomial.1 <- function(n, p) {
        k <- length(p)
        tabulate(sample(k, n, replace = TRUE, prob = p), nbins = k)
    }
    n <- rep(n, length = rows)
    p <- p[rep(1:nrow(p), length = rows), , drop = FALSE]
    t(apply(matrix(1:rows, ncol = 1), 1, function(i) rmultinomial.1(n[i], p[i, ])))
   # could be replaced by
   # sapply(1:rows, function(i) rmultinomial.1(n[i],  p[i, ]))
}
```

Because of the difference in scoping rules between the two implementations of S (i.e., R uses lexical scooping and S-PLUS uses static scooping), the function `rmultinomial` in R cannot just be sourced into S-PLUS. One alternative is to use the following S-PLUS implementation, which avoids defining the `rmultinomial.1` function, the source of the scoping problem above.

```
rmultinomial<-function (n, p, rows = max(c(length(n), nrow(p))))
{
    n <- rep(n, length = rows)
    p <- p[rep(1:nrow(p), length = rows), , drop = FALSE]

    sapply(1:rows,function(i,n,p) {
          k <- length(p[i,])
          tabulate(sample(k, n[i], replace = TRUE, prob = p[i,]), nbins = k)
          },n=n,p=p)
}
```

See Gentleman and Ilhaka (2000) or the R-FAQ for more information on the differences in scoping rules between R and S-PLUS.

Jim Lindsey's `gnlm` package contains a function called `fit.dist`, which fits a probability distribution of choice to frequency data. One can also use the `sample` function to generate (with or without replacement) from multiple categories given a set of probabilities for those categories.

More information on the use of these functions can be found in the S-PLUS or R online manuals or on pages 107-108 of Venables and Ripley (2002).

C. Proportion of Vegetarians (Statistical Inference for Binomial Parameters)

An example of the use of the S-PLUS distribution functions comes from computing the asymptotic and exact confidence intervals on the proportion of vegetarians (p. 16). In a questionnaire given to an introductory statistics class (n = 25), zero students said they were vegetarians. Assuming that the number responding yes is distributed binomially with success probability π, what is a 95% confidence interval for π, given that a point estimate, $\hat{\pi}$, is 0? Agresti cites three approximate methods and two exact methods. The approximate methods are given first.

1. Approximate Confidence Intervals on π

1) Inverting the Wald Test (AKA Normal Approximation)

$$\hat{\pi} \pm z_{\alpha/2}\sqrt{\frac{\hat{\pi}(1-\hat{\pi})}{n}} \tag{1.1}$$

which is computed quite easily in S-PLUS "by hand".

```
phat <- 0
n <- 25
phat + c(-1, 1) * qnorm(p = 0.975) * sqrt((phat * (1 - phat))/n)
[1] 0 0
```

However, its value is available via the `binconf` function in the Hmisc library in both S-PLUS and R, using the option `method="asymptotic"`.

```
library(Hmisc, T)
binconf(x=0, n=25, method="asymptotic")

PointEst Lower Upper
       0     0     0
```

2) The score confidence interval contains the π_0 values for which $|z_S| < z_{.025}$, where $z_S = (\hat{\pi} - \pi_0)\Big/\sqrt{\pi_0(1-\pi_0)/n}$. The endpoints of the interval solve the equations

$$(\hat{\pi} - \pi_0)\Big/\sqrt{\pi_0(1-\pi_0)/n} = \pm z_{.025} \tag{1.2}$$

Agresti gives the analytical expressions for these endpoints on his p. 16, which we could type into S-PLUS or R to get the values. However, even if there were no analytical expression, or we didn't want to try to find them, we could use the function `nlmin` to solve the set of simultaneous equations in (1.2), yielding an approximate confidence interval. (See point 3) and Chapter 3 for examples).

Built-in functions that compute the score confidence interval include the `prop.test` function (S-PLUS and R)

```
res<-prop.test(x=0,n=25,conf.level=0.95,correct=F)
res$conf.int

[1] 0.0000000 0.1331923
attr(, "conf.level"):
[1] 0.95
```

and the `binconf` function from `Hmisc` via its `method="wilson"` option:

```
library(Hmisc, T)
binconf(x=0, n=25, alpha=.05, method="wilson")

 PointEst         Lower     Upper
        0 1.202937e-017 0.1331923
```

3) A 95% likelihood-ratio (LR) confidence interval is the set of π_0 for which the likelihood ratio test has a p-value exceeding the significance level, α. The expression for the LR statistic is simple enough so that we can find the upper confidence bound analytically. However, we could have obtained it using `uniroot` (both R and S-PLUS) or using `nlmin` (S-PLUS only). These functions can be used to solve an equation, i.e., find its zeroes. `uniroot` is only for univariate functions. `nlmin` can also be used to solve a system of equations. A function doing the same in R would be function `nlm` or function `nls` (for nonlinear least squares) in library `nls`.

Using `uniroot` we set up the LR statistic as a function, giving as first argument the parameter for which we want the confidence interval. The remaining arguments are the observed successes, the total, and the significance level. `uniroot` takes an argument called `interval` that specifies where we want to search for the root. There cannot be a singularity at either of these values. Thus, we cannot use 0 and 1.

```
LR <- function(pi.0, y, n, alpha) {
          -2*(y*log(pi.0) + (n-y)*log(1-pi.0)) - qchisq(1-alpha,df=1)
       }

uniroot(f=LR, interval=c(0.000001,.999999), n=25, y=0, alpha=.05)

$root:
[1] 0.07395187

$f.root:
[1] -5.615436e-006
```

The function `nlmin` can be used to solve the nonlinear equation

$$-50\log(1-\pi_0) = \chi_1^2(0.05)$$

for π_0. We can take advantage of the function `solveNonlinear` listed in the help page for `nlmin` in S-PLUS. This function minimizes the squared difference between the LR statistic and the chi-squared quantile at α.

```
solveNonlinear <- function(f, y0, x, ...){
  # solve f(x) = y0
  # x is vector of initial guesses, same length as y0
  # ... are additional arguments to nlmin (not to f)
  g <- function(x, y0, f) sum((f(x) - y0)^2)
  g$y0 <- y0   # set g's default value for y0
  g$f <- f     # set g's default value for f
  nlmin(g, x, ...)
}

LR <- function(x) -50*log(1-x)       # define the LR function

solveNonlinear(f=LR, y0= qchisq(.95, df=1), x=.5)     # start finding the solution at
0.5

$x:
[1] 0.07395197

$converged:
```

```
[1] T

$conv.type:
[1] "x convergence"
```

2. Exact Confidence Intervals on π

There are several functions available for computing exact confidence intervals in R and S-PLUS. In R, the function `binom.test` computes a Clopper-Pearson interval. But, the same function in S-PLUS doesn't give confidence intervals, at least not automatically.

```
binom.test(x=0, n=25, conf.level=.95)   # R

        Exact binomial test

data:  0 and 25
number of successes = 0, number of trials = 25, p-value = 5.96e-08
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.0000000 0.1371852
sample estimates:
probability of success
                     0
```

The function `binconf` in the Hmisc library also gives Clopper-Pearson intervals via the use of the "`exact`" method.

```
library(Hmisc, T)
binconf(x = 0, n = 25, alpha = .05, method = "exact") # SPLUS

 PointEst Lower     Upper
        0     0 0.1371852
```

In addition, several individuals have written their own functions for computing Clopper-Pearson as well as other types of approximate intervals besides the normal approximation interval. A search of "exact binomial confidence interval" on the "S-news" search page (http://lib.stat.cmu.edu/cgi-bin/iform?SNEWS) gave several user-made functions.

The improved confidence interval described in Blaker (2000, cited in Agresti) can be computed in S-PLUS using the following function, which is a slight modification of the function appearing in the Appendix of the original reference.

```
acceptbin<-function(x, n, p){
   # computes the acceptability of p when x is observed and X is Bin(n, p)
   # adapted from Blaker (2000)
   p1<-1-pbinom(x-1, n, p)
   p2<-pbinom(x, n, p)
   a1<-p1 + pbinom(qbinom(p1, n, p) - 1, n, p)
   a2<-p2 + 1 - pbinom(qbinom(1-p2, n, p), n, p)
   min(a1, a2)
}


acceptinterval<-function(x, n, level, tolerance=1e-04){
   # computes acceptability interval for p at 1 - alpha equal to level
   # (in (0,1)) when x is an observed value of X which is Bin(n, p)
   # adapted from Blaker (2000)

   lower<-0; upper<-1

   if(x) {
```

```
        lower<-qbeta((1-level)/2, x, n-x+1)
        while(acceptbin(x, n, lower) < (1-level)) lower<-lower+tolerance
    }
    if(x!=n) {
        upper<-qbeta(1-(1-level)/2, x + 1, n - x)
        while(acceptbin(x, n, upper) < (1-level)) upper<-upper-tolerance
    }
    c(lower=lower, upper=upper)
}

acceptinterval(x=0, n=25, level=.95)

lower     upper
     0 0.1275852
```

D. The Mid-P-Value

A confidence interval can be based on the mid-p-value discussed in Section 1.4.5 of Agresti. For the Vegetarian example above, we can obtain a $100(1-\alpha)\%$ Clopper-Pearson confidence interval on π using the mid-p-value by finding all values of π_0 for which

$$\tfrac{1}{2}P(y=k|\pi_0)+P(y<k|\,\pi_0)>\alpha/2$$

and

$$\tfrac{1}{2}P(y=n-k|\pi_0)+P(y>n-k|\,\pi_0)>\alpha/2$$

where $P(y=k|\pi_0)$ is the binomial probability mass function, $y = 0$, and $n = 25$. For the example, this set of inequalities reduces to the inequality

$$\pi_0<1-\alpha^{1/n}=1-.05^{1/25}=0.113$$

because the lower limit is 0 when y = 0 (p. 18, Agresti).

E. Pearson's Chi-Squared Statistic

In both S-PLUS and R, one can find functions that will compute Pearson's Chi-Squared statistic. However, they appear in different places across the two implementations. In R, the `ctest` library contains the function `chisq.test`, which takes as arguments a set of frequencies, `x`, and its corresponding null probabilities, `p`. On Mendel's results (p. 22, Agresti) we get

```
library(ctest)
chisq.test(x=c(6022,2001),p=c(.75,.25))

        Chi-squared test for given probabilities

data:  c(6022, 2001)
X-squared = 0.015, df = 1, p-value = 0.9025
```

In S-PLUS, there does exist a built-in function called `chisq.test`, but its arguments are different, and the above code will not work. However, it calls a function `.pearson.X2` (as does the function `chisq.gof`) which allows one to input observed and expected frequencies.

```
res<-.pearson.x2(observed=c(6022,2001),expected=c(8023*.75,8023*.25))
1-pchisq(res$X2,df=1)

[1] 0.902528
```

The S-PLUS function `chisq.test` is used for situations where one has a 2x2 cross-tabulation (preferably computed using the function `table`) or two factor vectors, and wants to test for independence of the row and column variables.

F. Likelihood Ratio Chi-Squared Statistic

The likelihood ratio chi-squared statistic is easily computed "by hand" for multinomial frequencies. Using vectorized calculations, G^2 is for Mendel's data

```
obs <- c(6022, 2001)
expected <- 8023 * c(0.75, 0.25)
1-pchisq(2 * sum(obs * log(obs/expected)), df=1)

[1] 0.9025024
```

These methods can also be used on the dairy calves example on p. 25.

G. Maximum Likelihood Estimation

S-PLUS and R come with several functions for general optimization. We already saw examples of the use of `uniroot` for finding a zero of a univariate function and `nlmin` for minimizing a sum of squares. Here we look at some simple examples of using general optimization functions for maximum likelihood estimation with categorical data. Venables and Ripley (2002, Chapter 16) discuss these methods in more detail and give a few guidelines about parameterizing the objective function.

Lloyd (1999) gives data on the number of serious (fatal/nonfatal) accidents in the state of Victoria in Australia in 1985, separated by age group (over and under 21 years). Thus, we have a 2 x 2 contingency table with variables age and seriousness of accident. Lloyd initially treats the four accident counts as independent Poisson random variables with means λ_1, λ_2, λ_3, and λ_4, then considers several sub-models. One of the sub-models uses the (fictitious) information that 23.3% of serious accidents are expected to be fatal. This information gives the restriction $\phi_1/(\phi_1+\phi_2) = 0.233$, where $\phi_1 = \lambda_1 + \lambda_3$, the mean number of fatalities and $\phi_2 = \lambda_2 + \lambda_4$, the mean number of nonfatalities. The expected counts can be given by $e_1 = \lambda_1$, $e_2 = \lambda_2$, $e_3 = \phi_1 - \lambda_1$, and $e_4 = 3.292\phi_1 - \lambda_2$. Thus, this submodel has three parameters to estimate. The likelihood is (equation 1.10 in Lloyd)

$$\ell(\lambda_1, \lambda_2, \phi_1) = -4.292\phi_1 + y_1 \log \lambda_1 + y_2 \log \lambda_2 + y_3 \log(\phi_1 - \lambda_1) + y_4 \log(3.292\phi_1 - \lambda_2) \qquad (1.3)$$

The observed values of the counts are $y_1 = 11$, $y_2 = 62$, $y_3 = 4$, and $y_4 = 7$. The function `nlminb` in S-PLUS finds a local minimum of a twice-differentiable function possibly subject to boundary constraints on the parameters. A gradient function and Hessian function can be supplied as arguments. The algorithm used is a quasi-Newton method if a Hessian is not supplied and Newton's method if it is. If no gradient is supplied, a finite difference approximation is used. Lloyd gives the first derivatives on p. 11, but first we will try the estimation without analytically supplied derivatives. Of course, we must remember to use the negation of (1.3) as the objective function to minimize (a VERY common mistake by me).

Here I set the objective function (with first argument the vector of parameters) and then send it into `nlminb`, along with starting values and upper and lower bounds on each parameter. The starting values were chosen as $\lambda_i^{(0)} = y_i$ (i = 1, 2) and $\phi_1^{(0)} = 2(y_1 + y_3)$ to ensure that the logs are positive at the initial values. To help prevent warnings about NAs generated in log(x), I have scaled the step-length of

the ϕ_1 parameter using the `scale` argument so that when p[1] is subtracted from p[3], we should always get a positive value. I have also started ϕ_1 at a value much larger than λ_1.

```
obj.function<-function(p, y){
    -(-4.292*p[3] + y[1]*log(p[1]) + y[2]*log(p[2]) + y[3]*log(p[3]-p[1]) +
    y[4]*log(3.292*p[3]-p[2]))
}

nlminb(start=c(11, 62, 2*(11+4)), obj.function, lower=c(0,0,0), upper=c(Inf, Inf,
    Inf), scale=c(1,1,10), y=c(11, 62, 4, 7))

$parameters:
[1] 14.35228 57.89246 19.57129

$objective:
[1] -216.6862

$message:
[1] "RELATIVE FUNCTION CONVERGENCE"

$grad.norm:
[1] 0.00001071819

$iterations:
[1] 22
```

…snip

We get convergence (relative function convergence) in 22 iterations. The parameter estimates are almost identical to those given by Lloyd.

Now, if we wanted to supply first derivatives, we can get them using the `deriv` function, which returns the function along with an attribute that is a functional representation of the gradient. Or, we could supply the gradient as a separate function that returns three values when evaluated (for the three elements of the gradient). In this case the objective function is simple enough for `deriv` to handle. We make some modifications first, however.

```
obj.res<-deriv(~-(-4.292*pi1 + y1*log(la1) + y2*log(la2) + y3*log(pi1-la1) +
    y4*log(3.292*pi1-la2)),
    c("la1","la2","pi1"),
    function(la1, la2, pi1, y1, y2, y3, y4) NULL)

obj.gr<-function(p, y){
    la1<-p[1]; la2<-p[2]; pi1<-p[3]; y1<-y[1]; y2<-y[2]; y3<-y[3]; y4<-y[4]
    attr(obj.res(la1, la2, pi1, y1, y2, y3, y4), "gradient")
}

nlminb(start=c(11, 62, 2*(11+4)), objective=obj.function, gradient=obj.gr,
    lower=c(0,0,0), upper=c(Inf, Inf, Inf), scale=c(1,1,10), y=c(11, 62, 4, 7))


$parameters:
[1] 14.35228 57.89246 19.57129

$objective:
[1] -216.6862

$message:
[1] "RELATIVE FUNCTION CONVERGENCE"

$grad.norm:
```

```
[1] 5.191676e-007

$iterations:
[1] 22
```

We get the same result. One could supply the Hessian using the function `deriv3`. The Mass library supplies a function called `vcov.nlminb` to extract the inverse of observed information (or a finite difference approximation if the Hessian is not supplied) from an `nlminb` object.

In R, we can use the function `nlm`, which is similar in that it minimizes an objective function of several variables, allows boundary constraints, and uses a quasi-Newton optimizer. However, it does not allow box constraints on the parameters.

```
obj.res<-deriv(~-(-4.292*pi1 + y1*log(la1) + y2*log(la2) + y3*log(pi1-la1) +
      y4*log(3.292*pi1-la2)),
    c("la1","la2","pi1"),
    function(la1, la2, pi1, y1, y2, y3, y4) NULL)

obj.function<-function(p, y){
    value<--(-4.292*p[3] + y[1]*log(p[1]) + y[2]*log(p[2]) + y[3]*log(p[3]-p[1]) +
             y[4]*log(3.292*p[3]-p[2]))
    la1<-p[1]; la2<-p[2]; pi1<-p[3]; y1<-y[1]; y2<-y[2]; y3<-y[3]; y4<-y[4]
    attr(value, "gradient")<-attr(obj.res(la1, la2, pi1, y1, y2, y3, y4),
             "gradient")
    value
}

nlm(f=obj.function, p=c(11, 62, 2*(11+4)), y=c(11, 62, 4, 7), typsize=c(1,1,.10))

$minimum
[1] -216.6862

$estimate
[1] 14.35229 57.89247 19.57130

$gradient
[1]  1.276756e-06 -1.085920e-07 -4.459707e-09

$code
[1] 1

$iterations
[1] 18
```

Again, we use the argument `typsize` to control warnings about NAs in log(x).

The function `optim` includes several other algorithms besides Newton-type methods (including simulated annealing) and allows box constraints.

```
obj.function<-function(p, y){
    -(-4.292*p[3] + y[1]*log(p[1]) + y[2]*log(p[2]) + y[3]*log(p[3]-p[1]) +
    y[4]*log(3.292*p[3]-p[2]))
}

obj.res<-deriv(~-(-4.292*pi1 + y1*log(la1) + y2*log(la2) + y3*log(pi1-la1) +
              y4*log(3.292*pi1-la2)),
       c("la1","la2","pi1"),
       function(la1, la2, pi1, y1, y2, y3, y4) NULL)


obj.gr<-function(p, y){
    la1<-p[1]; la2<-p[2]; pi1<-p[3]; y1<-y[1]; y2<-y[2]; y3<-y[3]; y4<-y[4]
```

```
    attr(obj.res(la1, la2, pi1, y1, y2, y3, y4), "gradient")
}

optim(par=c(11, 62, 2*(11+4)), fn=obj.function, gr=obj.gr, lower=c(0,0,0),
      method="L-BFGS-B", control=list(parscale=c(1,1,10)), y=c(11, 62, 4, 7))

$par
[1] 14.35235 57.89246 19.57130

$value
[1] -216.6862

$counts
function gradient
      25       25

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

The Mass library for S-PLUS and R also includes a version of `optim`. And, it contains a function called `fitdistr` for univariate maximum likelihood estimation.

Chapter 2: Describing Contingency Tables

A. Summary of Chapter 2, Agresti

Chapter two in Agresti introduces two-way $I \times J$ contingency tables. If both the row and column of a table denote random variables, then the probabilities $\{\pi_{ij}\}$ define the joint distribution of the two variables. The marginal distributions are denoted by $\{\pi_{i+}\}$ for the row variable and $\{\pi_{+j}\}$ for the column variable. For a fixed value i of the row variable, the column variable has the conditional distribution $\{\pi_{1|i},...,\pi_{J|i}\}$. The conditional distribution is especially important if the row variable is fixed by design and not free to vary for each observation.

With diagnostic tests for a disease, the *sensitivity* of the test is the conditional probability that the diagnostic test is positive given that subject has the disease. The *specificity* of the test is the conditional probability that the test is negative given that the subject does not have the disease. In a 2x2 table with rows indicating disease status (yes, no) and columns indicating test result (positive, negative), the sensitivity is $\pi_{1|1}$, and the specificity is $\pi_{2|2}$.

Row and column variables are independent if the conditional distribution of the column variable given the row variable is the same as the marginal distribution of the column variable (and vice versa). That is, $\pi_{j|i} = \pi_{+j}$ for $i = 1,\ldots, I$, and $\pi_{i|j} = \pi_{i+}$ $j = 1,\ldots, J$. Equivalently, if all joint probabilities equal the product of their marginal probabilities: $\pi_{ij} = \pi_{i+}\pi_{+j}$, for all i and j. Thus, when the two variables are independent, knowledge of the value of the row variable does not change the distribution of the column variable, and vice versa.

When the row variable is an explanatory variable and the column is a response variable, then there is no joint distribution, and independence is referred to as homogeneity of the conditional distributions of the column variable given a value for the row variable.

The distributions of the cell counts $\{Y_{ij}\}$ differ depending on how sampling was done. If observations are to be collected over a certain period of time and cross-classified into one of the $I \times J$ categories, then a Poisson sampling model might be used where cell counts are treated as independent Poisson random variables with parameters $\{\mu_{ij}\}$. If the total sample size of observations is fixed in advance (e.g., in a *cross-sectional* study), then a multinomial sampling model might be used where cell counts are treated as multinomial random variables with index n and probabilities $\{\pi_{ij}\}$. If the row totals are fixed in advance, perhaps as fixed-size random samples drawn from specific populations that are to be compared, as in *prospective* studies, then a product multinomial sampling model may apply where for each i, the counts $\{Y_{j|i}\}$ have a multinomial distribution with index n_i and probabilities $\pi_{j|i}$ $j = 1,\ldots, J$. If both row and column totals are fixed by design, then a hypergeometric sampling distribution applies for the cell counts.

However, there are times when certain sampling models are assumed, but sampling was actually done differently. For example, when the row variable is an explanatory variable, product multinomial sampling model may be used even though the row totals were not actually fixed in advance. Also, the Poisson model is used even when the total sample size is fixed in advance (see Chapter 8 of Agresti).

Section 2.2 discusses comparing two proportions from two samples, including the difference of proportions, relative risk, and odds ratio. The relative risk compares the proportions of the two groups in a ratio. If the rows of a 2x2 table represent groups and the columns represent a binary response, then the relative risk of a positive response is the ratio $\pi_{1|1}/\pi_{1|2}$. A relative risk of 1.0 corresponds to independence. A relative risk of C means that $\pi_{1|1} = C\pi_{1|2}$. The odds ratio is the ratio of odds of a positive response by group

$$\theta = \frac{\pi_{1|1}/(1-\pi_{1|1})}{\pi_{1|2}/(1-\pi_{1|2})} = \frac{\pi_{11}\pi_{22}}{\pi_{12}\pi_{21}} \quad (2.1)$$

When $\theta = 1$, the row and column variables are independent. Values of θ farther from 1.0 in a given direction represent stronger association. For example, as on p. 45 of Agresti, when $\theta = 0.25$, the odds of "success" in group 1 (row 1) are 0.25 times the odds in group 2 (row 2), or equivalently, the odds of success in group 2 are 4 times the odds in group 1. The odds ratio can be used with a joint distribution of the row and column variables too. Indeed, it can be used with prospective (rows totals fixed), retrospective (column totals fixed), and cross-sectional designs. Finally, if the rows and columns are interchanged, the value of the odds ratio does not change. The sample odds ratio uses the observed sample counts, n_{ij}.

Odds ratios can be computed for retrospective sampling designs (case-control studies). Relative risk cannot be computed because the outcome variable is fixed by design. However, if the probability of the outcome is close to zero for both groups, then the odds ratio can be used to approximate relative risk using the formula on p. 47 of Agresti.

In observational studies, confounding variables can be controlled with stratification or conditioning. The association between two variables X and Y given that another measured variable Z takes the value *z* is called a conditional association. The 2 x 2 table resulting from cross-classifying all observations with Z = *z* by their X and Y values is called a *partial table*. If Z is ignored, the X-Y table is called a *marginal table*. Simpson's Paradox is the result that a marginal association can have a different direction than a conditional association. For example, in the death penalty example on p. 49-51, ignoring victim's race, the death penalty (Y) is more likely for whites than for blacks (X). However, conditioning on victim's race (either black or white), the death penalty is more likely for blacks than for whites. The paradox in this case can be explained by the strong association between victim's race (ignored in the marginal association) and defendant's race and that between victim's race and the death penalty. The death penalty was more likely when the victims were white (regardless of defendant race). Also, whites were more likely to kill whites than any other victim/defendant race combination in the sample. So, there are a lot of whites receiving the death penalty in the sample. On the other hand, blacks were more likely to kill blacks. Thus, there are fewer blacks receiving the death penalty in the sample. But, if we look at only white victims, there are relatively more blacks receiving the death penalty than whites. The same is true for black victims. An unmodeled association between victim's and defendant's race hides this conclusion.

Does Simpson's Paradox imply that we should distrust all contingency table analyses? After all, there are undoubtedly unmeasured variables that could be potential conditioning variables in all contingency tables. Could these variables change the direction of marginal associations? Page 51 in Agresti paraphrases J. Cornfield's result "that with a very strong XY association [marginal association], a very strong association must exist between the confounding variable Z and both X and Y in order for the effect to disappear or change …".

For I x J x K tables (where X has I levels, Y has J levels, and Z has K levels), if X and Y are independent in partial table *k*, then X and Y are conditionally independent given that Z takes on value *k*. If X and Y are independent at all levels of Z, then X and Y are conditionally independent given Z. Or, X and Y only depend on each other through Z. Once variability in Z is removed, by fixing it, X and Y are no longer related statistically. Conditional independence does not imply marginal independence. For 2 x 2 x K tables, X and Y are conditionally independent given Z if the odds ratio between X and Y equals 1 at each category of Z. For the general case of I x J x K tables, independence is equivalent to all $(I-1)(J-1)$ *local odds ratios* equaling 1.0.

An analogy to no three-way interaction in ANOVA is *homogeneous association*. A 2 x 2 x K table has homogeneous XY association if the conditional odds ratios comparing two categories of X to two categories of Y are the same at each level of Z. When interaction exists, the conditional odds ratio for any pair of variables (say X and Y) changes across categories of the third (say Z), wherein the third variable is called an *effect modifier* because the effect of X on Y (the response) changes depending on the level of Z. For the general case of I x J x K tables, homogeneous XY association means that any conditional odds ratio formed using two categories of X and two categories of Y is the same at each category of Z.

The chapter concludes with discussion of summary measures of association for nominal and ordinal data. The measures described include Kendall and Stuart's measure of proportional reduction in variance from the marginal distribution of the response to the conditional distributions given the value of an explanatory vector, Theil's uncertainty coefficient, the proportional reduction in entropy (or uncertainty) in response given explanatory variables, and measures for ordinal association such as concordance and Gamma.

B. Comparing two proportions

The Aspirin and Heart Attacks example is used to illustrate several ways to compare proportions between two groups. The two groups are aspirin-takers and placebo-takers. Fatal and non-fatal heart attacks are not distinguished. The data are in Table 2.1 on p. 37 of Agresti. Setting up the data is easy:

```
x<-c(104,189) # aspirin, placebo
n<-c(11037,11034)
```

Then, to test H_0:p_1=p_2 (equal probabilities of heart attack per group), one can use the `prop.test` function. The output given here is from S-PLUS.

```
prop.test(x, n)

    2-sample test for equality of proportions with continuity correction

data:  x out of n

X-square = 24.4291, df = 1, p-value = 0

alternative hypothesis: two.sided

95 percent confidence interval:
 -0.010814914 -0.004597134

sample estimates:
 prop'n in Group 1 prop'n in Group 2
        0.00942285        0.01712887
```

The output from R is almost identical except that the p-value is stated as: `p-value = < 2.2e-16`. `X-square` is the value of the chi-squared statistic. One can choose not to use the continuity correction with `correct=F`.

We can obtain the p-value by extracting it from `prop.test`. Both S-PLUS and R give essentially 0 as the value here.

```
prop.test(x,n)$p.value
[1] 7.709708e-007
```

A one-sided test of the hypotheses, H_0:p_1=p_2 v. H_1:p_1<p_2, can be obtained using the `alternative` option:

```
prop.test(x,n,alt="less")$p.value
[1] 3.854854e-007
```

The proportions themselves can be extracted from the `estimate` component, which is a numeric vector of length two here. So, the sample difference of proportions is computed as:

```
temp<-prop.test(x,n)
names(temp$estimate)<-NULL  # optional
```

```
temp$estimate[1]-temp$estimate[2]
[1] -0.007706024
```

Other useful quantities are easily computed. Here, I calculate the relative risk and odds ratio using the object `temp` above, as well as the original data vectors:

Relative risk:

```
temp$estimate[2]/temp$estimate[1]
[1] 1.817802
```

Odds ratio (simple enough not to need `temp`):

```
x[2]*(n[1]-x[1])/(x[1]*(n[2]-x[2]))
[1] 1.832054
```

C. Partial Association in Stratified 2 x 2 Tables

The Death Penalty example is given on p. 48 of Agresti to illustrate partial tables and conditional odds ratios. The effect of the defendant's race (X) on the death penalty verdict (Y) is studied, treating the victim's race (Z) as a control variate. The 2 x 2 x 2 table can be created using the function `crosstabs` in S-PLUS (`xtabs` in R). The function `print.crosstabs` can also be called directly with output from `crosstabs` to control what is actually printed after the call.

```
vic.race<-c("white","black")
def.race<-vic.race
death.penalty<-c("yes", "no")
datalabel<-list(defendant=def.race,death=death.penalty,victim=vic.race)
table.2.6<-fac.design(c(2,2,2),factor.names=datalabel)   # sets up the combinations of
   the levels as a factorial design, using labels datalabel
data<-c(53, 11, 414, 37, 0, 4, 16, 139)
table.2.6<-cbind(table.2.6,count=data)
crosstabs(count~defendant+death+victim ,data=table.2.6)

crosstabs(formula = count ~ defendant + death + victim, data = table.2.6)
674 cases in table
+----------+
|N         |
|N/RowTotal|
|N/ColTotal|
|N/Total   |
+----------+
victim=white
defendant|death
         |yes    |no     |RowTotl|
---------+-------+-------+-------+
white    | 53    |414    |467    |
         |0.11   |0.89   |0.91   |
         |0.83   |0.92   |       |
         |0.079  |0.61   |       |
---------+-------+-------+-------+
black    | 11    | 37    |48     |
         |0.23   |0.77   |0.093  |
         |0.17   |0.082  |       |
         |0.016  |0.055  |       |
---------+-------+-------+-------+
ColTotl  |64     |451    |515    |
         |0.12   |0.88   |       |
---------+-------+-------+-------+
```

```
victim=black
defendant|death
         |yes     |no      |RowTotl|
---------+--------+--------+--------+
white    |   0    |  16    |16      |
         |0       |1       |0.1     |
         |0       |0.1     |        |
         |0       |0.024   |        |
---------+--------+--------+--------+
black    |   4    |139     |143     |
         |0.028   |0.97    |0.9     |
         |1       |0.9     |        |
         |0.0059  |0.21    |        |
---------+--------+--------+--------+
ColTotl  |4       |155     |159     |
         |0.025   |0.97    |        |
---------+--------+--------+--------+
Test for independence of all factors
    Chi^2 = 419.5589 d.f.= 4 (p=0)
    Yates' correction not used
    Some expected values are less than 5, don't trust stated p-value
```

The differences in R are enough so that the above is worth repeating in R. Below is the same result (including set up) from R's `xtabs`. Note that the `fac.design` has been replaced by `expand.grid`. The output is also much more terse by default.

```
vic.race<-c("white","black")
def.race<-vic.race
death.penalty<-c("yes", "no")
datalabel<-list(defendant=def.race,death=death.penalty,victim=vic.race)
table.2.6<- expand.grid(defendant=def.race,death=death.penalty,victim=vic.race)
data<-c(53, 11, 414, 37, 0, 4, 16, 139)
table.2.6<-cbind(table.2.6,count=data)
xtabs(count~defendant+death+victim ,data=table.2.6)
```

```
, , victim = white

         death
defendant yes  no
    white  53 414
    black  11  37

, , victim = black

         death
defendant yes  no
    white   0  16
    black   4 139
```

The function `crosstabs` returns many results. One can extract the cell proportions via the `marginals` attribute.

```
temp<-crosstabs(count~defendant+death+victim ,data=table.2.6)
attr(temp,"marginals")$"N/RowTotal"  # return the cell proportions (Figure 2.1)
```

```
, , white
            yes       no
white 0.1134904 0.8865096
black 0.2291667 0.7708333

, , black
             yes       no
white 0.00000000 1.000000
black 0.02797203 0.972028
```

Getting the marginals can be done by re-calling the function without `victim`, or one can use the `update` function to update the call by subtracting `victim` from the formula. `update` is a generic function for which methods are written for different classes of objects. There is no method for class `crosstabs`, so a call to `update` via the set of commands:

```
temp<-crosstabs(count~defendant+death+victim, data=table.2.6)
update(object=temp, formula=~ . -victim)
```

will cause S-PLUS to use the default method, `update.default`, to re-evaluate the call. However, this method is not suitable for objects of class `crosstabs`. Thus, we can create our own method, which turns out to change just one line of `update.default` (else I probably wouldn't have it in here!). Creating a method is equivalent operationally to creating a function. Thus, the same tools that work to create functions work to create methods (which are just functions, themselves). See the Introduction to this manual for how to edit functions in S-PLUS 6.1 and R.

First, I define a method function called `update.crosstabs`, which is the same code as `update.default`, but changes the line in `update.default`

```
if(!missing(formula)) newcall$formula<-as.vector(update.formula(object, formula, evaluate = T))
```

to

```
if(!missing(formula))newcall$formula<-as.vector(my.update.formula(object, formula, evaluate = T))
```

where the function `my.update.formula` changes the function `update.formula` by substituting the line

```
form <- as.formula(object)
```

with

```
form <- as.formula(attr(object, "call"))
```

Then, I set the update method for `crosstabs` to be `update.crosstabs`

```
setMethod("update","crosstabs",update.crosstabs)
```

The call

```
update(object=temp, formula=~ . -victim)
```

gives

```
Call:
crosstabs(formula = count ~ defendant + death, data = table.2.6)
674 cases in table
+----------+
|N         |
|N/RowTotal|
|N/ColTotal|
|N/Total   |
+----------+
defendant|death
         |yes    |no     |RowTotl|
---------+-------+-------+-------+
white    | 53    |430    |483    |
         |0.11   |0.89   |0.72   |
         |0.78   |0.71   |       |
```

```
        |0.079  |0.64   |       |
--------+-------+-------+-------+
black   | 15    |176    |191    |
        |0.079  |0.92   |0.28   |
        |0.22   |0.29   |       |
        |0.022  |0.26   |       |
--------+-------+-------+-------+
ColTotl|68      |606    |674    |
        |0.1    |0.9    |       |
--------+-------+-------+-------+
Test for independence of all factors
    Chi^2 = 1.468519 d.f.= 1 (p=0.2255796)
    Yates' correction not used
```

which is the bottom portion of Table 2.6 in Agresti.

In R, we also create an `update` method for `xtabs`, which is a function that we will call `update.xtabs`. This will be the same as `update.default`, with the following substitutions:

Change the line

```
call <- object$call
```

to

```
call<-attr(object, "call")
```

And, change the line

```
call$formula <- update.formula(formula(object), formula.)
```

to

```
call$formula <- update.formula(call$formula, formula.)
```

The method has been constructed, as verified by a call to **`methods("update")`** or to **`methods(class = xtabs)`**. Note that this procedure uses the S3 scheme (see the help file), as opposes to the S4 scheme used for the S-PLUS example.

Then, the `update` call gives the marginal table

```
update(temp, formula= ~.-victim)
```

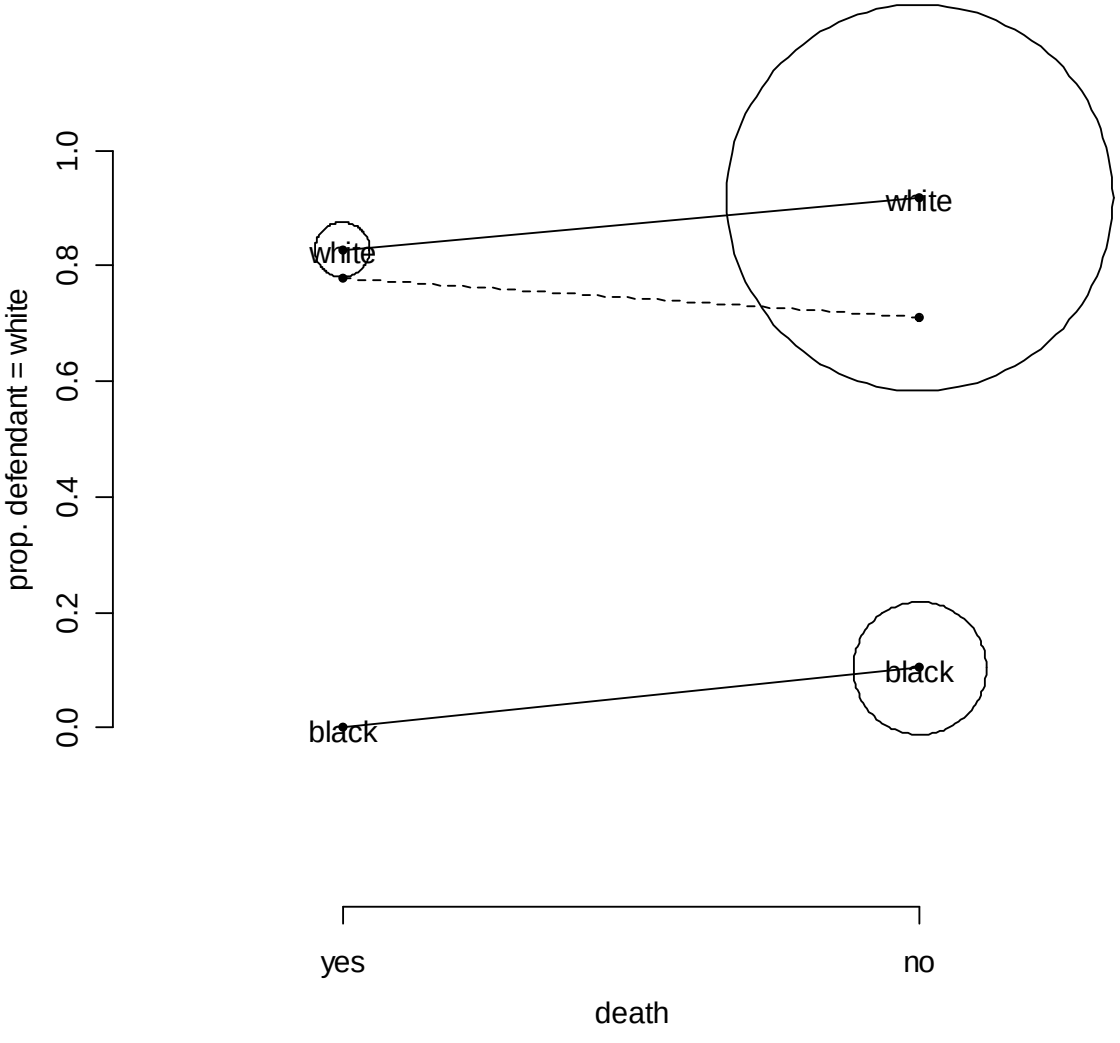
```
         death
defendant yes  no
    white  53 430
    black  15 176
```

Paik Diagram

Tobias Verbeke has written a function that draws the Paik Diagram (Paik, 1985, citation in Agresti), which is a graphical representation of Simpson's Paradox. The diagram is similar to Figure 2.2 in Agresti (p. 50) for the death penalty example. However, in the example sent to me, Verbeke plots Death Penalty by proportion white defendant. From this plot, you can see the paradox in a different way.

The function code is given in the file that holds the R code for this manual.

```
paik(xtabs(count~defendant+death+victim ,data=table.2.6))
```



In the body of the plot are the segments for white and black victims. The sizes of the circles connected by segments represent the relative count at that particular cell of the table. The dotted segment represents the marginal table (not conditioning on race of victim). It shows that the proportion of white defendants is higher for the death penalty group than for those who did not get the death penalty.

From conditional associations, however, we see that a white victim is much more likely to have a white defendant than a black defendant (due to the height of the "white" segment), and a black victim is much more likely to have a black defendant. Those who receive the death penalty are more frequently those who had white victims than those who had black victims (see the circles for the "yes" column).

D. Conditional Odds Ratios

As the objects returned by `xtabs` in R and `crosstabs` in S-PLUS are already arrays, we can just use `apply` on the 2D slices representing the conditional tables.

```
apply(temp,3,function(x) x[1,1]*x[2,2]/(x[2,1]*x[1,2]))
```

```
    white     black
0.4306105 0.0000000
```

The R package `vcd` has a function `oddsratio`, which computes conditional odds ratios for 2x2x… tables, along with asymptotic confidence intervals. If we use it on the death penalty table to get odds ratios within victim's race, we get

```
summary(oddsratio(temp, log=F, stratum=3))
```

```
      Odds Ratio        lwr        upr
white  0.4208843 0.20498745  0.8641678
black  0.9393939 0.04838904 18.2367947
```

Note that the odds ratio for black victims is noticeably larger than that computed using `apply`. The function `oddsratio` adds 0.5 to each count prior to computing the odds ratios. The estimate near 1.0 implies that for black victims, the odds of getting the death penalty are equal for white and black defendants.

A plot method is also available for `oddsratio` (`plot.oddsratio`), which may be quite useful with a large table.

E. Summary Measures of Assocation: Ordinal Trends

An example to illustrate ordinal measures of association comes from the income and job satisfaction data in Table 2.8 (p. 57, Agresti). The respondents are black males in the U.S. Four ordinal classifications constitute the response categories for both variables (see the table). We might postulate that as income increases, reported job satisfaction increases as well, and vice versa. Measures analogous to correlation measure the degree of monotonic relationship between the two variables. Agresti uses a concordance measure to quantify the association.

A pair of individuals in the sample is *concordant* if the individual who is ranked higher (among the two individuals) on one variable, say X, also ranks higher on the second variable, say Y. The pair is *discordant* if the individual ranking higher on X ranks lower on Y. The pair is tied if the individuals have the same classification on X and/or Y. If the number of concordant pairs, say C, exceeds the number of discordant pairs, say D, then a positive monotonic relationship is supported. A statistic that is based on this difference is Goodman and Kruskal's Gamma. It is the estimated difference between the probability of concordance and the probability of discordance, given that a pair is untied. Gamma, γ, is estimated as

$$\hat{\gamma} = \frac{(C-D)}{(C+D)} \tag{2.2}$$

Gamma measures monotonic association between two variables, with range $-1 \le \hat{\gamma} \le 1$. Positive and negative associations have the corresponding sign changes in $\hat{\gamma}$. Perfect monotonicity in the sample ($|\hat{\gamma}| = 1$) occurs when there are either no discordant pairs (D = 0: $\hat{\gamma} = 1$) or there are no concordant pairs (C = 0: $\hat{\gamma} = -1$).

One can create a cross-classified table in S-PLUS out of Table 2.8 using the following commands (use `xtabs` in R instead of `crosstabs`):

```
income<-c("<15000","15000-25000","25000-40000",">40000")
jobsat<-c("VD","LD","MS","VS")
table.2.8<-expand.grid(income=income,jobsat=jobsat)
data<-c(1,2,1,0,3,3,6,1,10,10,14,9,6,7,12,11)
table.2.8<-cbind(table.2.8,count=data)
(temp<-crosstabs(count~income+jobsat,table.2.8))
```

```
Call:
crosstabs(formula = count ~ income + jobsat, data = table.2.8)
96 cases in table
+----------+
|N         |
|N/RowTotal|
|N/ColTotal|
|N/Total   |
+----------+
income |jobsat
       |VD     |LD     |MS     |VS     |RowTotl|
-------+-------+-------+-------+-------+-------+
<15000 | 1     | 3     |10     | 6     |20     |
       |0.05   |0.15   |0.5    |0.3    |0.21   |
       |0.25   |0.23   |0.23   |0.17   |       |
       |0.01   |0.031  |0.1    |0.062  |       |
-------+-------+-------+-------+-------+-------+
15000-2| 2     | 3     |10     | 7     |22     |
       |0.091  |0.14   |0.45   |0.32   |0.23   |
       |0.5    |0.23   |0.23   |0.19   |       |
       |0.021  |0.031  |0.1    |0.073  |       |
-------+-------+-------+-------+-------+-------+
25000-4| 1     | 6     |14     |12     |33     |
       |0.03   |0.18   |0.42   |0.36   |0.34   |
       |0.25   |0.46   |0.33   |0.33   |       |
       |0.01   |0.062  |0.15   |0.12   |       |
-------+-------+-------+-------+-------+-------+
>40000 | 0     | 1     | 9     |11     |21     |
       |0      |0.048  |0.43   |0.52   |0.22   |
       |0      |0.077  |0.21   |0.31   |       |
       |0      |0.01   |0.094  |0.11   |       |
-------+-------+-------+-------+-------+-------+
ColTotl|4      |13     |43     |36     |96     |
       |0.042  |0.14   |0.45   |0.38   |       |
-------+-------+-------+-------+-------+-------+
Test for independence of all factors
    Chi^2 = 5.965515 d.f.= 9 (p=0.7433647)
    Yates' correction not used
    Some expected values are less than 5, don't trust stated p-value
```

Here is a function for computing Goodman and Kruskal's gamma. There is a different version of this function in Chapter 3 of this manual (called `Gamma2.f`). This version uses the computations from problem 3.27 in Agresti, and also computes the standard error of Gamma. It is faster than `Gamma.f`.

```
Gamma.f<-function(x)
{
    # x is a matrix of counts.  You can use output of crosstabs or xtabs.
    n<-nrow(x)
    m<-ncol(x)
    res<-numeric((n-1)*(m-1))
    for(i in 1:(n-1)) {
        for(j in 1:(m-1)) res[j+(m-1)*(i-1)]<-x[i,j]*sum(x[(i+1):n,(j+1):m])
    }
    C<-sum(res)
    res<-numeric((n-1)*(m-1))
    iter<-0
    for(i in 1:(n-1))
        for(j in 2:m) {
            iter<-iter+1; res[iter]<-x[i,j]*sum(x[(i+1):n,1:(j-1)])
        }
    D<-sum(res)
    gamma<-(C-D)/(C+D)
    list( gamma=gamma, C=C, D=D)
    }
```

We can use this on table 2.8 by just inputting the result of the `crosstabs` call above:

```
Gamma.f(temp)

$gamma:
[1] 0.2211009

$C:
[1] 1331

$D:
[1] 849
```

Selvin (1998) computes the number of concordant and discordant pairs using the `outer` function along with `ifelse` statements (Selvin, p. 339). However, the procedure is memory intensive. The function above takes between 0.33 and 0.60 CPU seconds on a Pentium 4, 1.4 GHz, with 512 MB RAM.

Other measures of association can be computed immediately from the chi-square value output from `chisq.test` (e.g., phi, Cramer's V, Cramer's C). See Selvin p. 336ff for more details. The R package `vcd` has a function `assoc.stats` that computes these association measures along with the Pearson and LR chi-square tests. For example, on the job satisfaction data (where we used `xtabs` instead of `crosstabs`),

```
summary(assoc.stats(temp))

             jobsat
income        VD LD MS VS
  <15000       1  3 10  6
  15000-25000  2  3 10  7
  25000-40000  1  6 14 12
  >40000       0  1  9 11

                    X^2 df P(> X^2)
Likelihood Ratio 6.7641  9  0.66167
Pearson          5.9655  9  0.74336

Phi-Coefficient   : 0.249
Contingency Coeff.: 0.242
Cramer's V        : 0.144
```

A nice method for an object of class `crosstabs` is the "`[`" method. This allows us to select smaller tables in the following way. Suppose I want to print Table 2.8 with the last category of the income variable eliminated. This is

```
temp[1:3,1:4]

Call:
crosstabs(formula = count ~ income + jobsat, data = table.2.8)
75 cases in table
+----------+
|N         |
|N/RowTotal|
|N/ColTotal|
|N/Total   |
+----------+
income |jobsat
       |VD     |LD     |MS     |VS     |RowTotl|
-------+-------+-------+-------+-------+-------+
<15000 | 1     | 3     |10     | 6     |20     |
       |0.05   |0.15   |0.5    |0.3    |0.27   |
```

	0.25	0.23	0.23	0.17	
	0.01	0.031	0.1	0.062	
15000-2	2	3	10	7	22
	0.091	0.14	0.45	0.32	0.29
	0.5	0.23	0.23	0.19	
	0.021	0.031	0.1	0.073	
25000-4	1	6	14	12	33
	0.03	0.18	0.42	0.36	0.44
	0.25	0.46	0.33	0.33	
	0.01	0.062	0.15	0.12	
ColTotl	4	12	34	25	75
	0.053	0.16	0.45	0.33	

Test for independence of all factors
   Chi^2 = 1.432754 d.f.= 6 (p=0.9638357)
   Yates' correction not used
Some expected values are less than 5, don't trust stated p-value

Chapter 3: Inference for Contingency Tables

A. Summary of Chapter 3, Agresti

This chapter discusses interval estimation and testing for two-way contingency tables, both unordered and ordered. Confidence intervals for association parameters like the odds ratio, difference in proportions, and relative risk for 2x2 tables can be computed using large-sample normality (Wolf's procedure). Score and profile likelihood confidence intervals are better alternatives, especially for smaller sample sizes.

For an *I* x *J* contingency table, the hypothesis that the row and column variables are independent may be tested using a chi-square test (either likelihood ratio statistic or Pearson's chi-squared statistic). Although the two statistics are asymptotically equivalent and asymptotically chi-squared, Pearson's chi-squared statistic tends to be better for smaller counts. If the row and column variables are ordered, then a trend may be used as the alternative hypothesis. Chi-squared tests can be followed up by residual analysis and chi-squared tests on partitioned tables to indicate exactly where association lies. Rules for partitioning likelihood ratio chi-squared tests appear on p. 84 of Agresti.

If independence of the two classification factors of an *I* x *J* contingency table approximately describes the table, then the MLEs of the cell probabilities under the hypothesis of independence can have lower MSE than can the MLEs under the alternative (i.e., the sample proportions), except for when the sample size is very large. This is because the variance of the MLEs under independence is smaller because they are based on fewer parameters. However, they are biased. The bias can dominate the variance when the sample size increases, thereby causing the independence MLEs to lose their advantage over the sample proportions. Agresti gives details on p. 85-86.

If row and column variables are ordered, then a test of independence that has a trend alternative has greater power to detect a true association (if the association is a trend of that type) than would a general-purpose chi-squared test. One reason for this is due to the fewer degrees of freedom for the trend statistics versus the chi-squared statistics, as there are fewer parameters to estimate for the trend statistics.

There is a disadvantage in using trend tests with ordinal level variables because one has to choose the scores for the levels of each of the variables. An inappropriate choice can downplay an association, making the test less sensitive to it. Agresti shows that the popular choice to use category midranks as scores can affect sensitivity of test statistics if the cell sizes are quite disparate. A suggestion is to use equally-spaced scores unless there is an inherent numerical order in the levels already.

When either the row or column variable has two categories, then the tests based on linear or monotone trend that are discussed in Section 3.4 of Agresti reduce to certain well-known nonparametric tests. See p. 90 in Agresti. However, some care is needed when using midranks for the categories of the response variable (reducing to Wilcoxon Mann-WhitneyTest), as there may be very many ties (see Sprent, 1997).

With a small total sample size, the exact tests of Section 3.5 are preferred over analogous large-sample tests. Fisher's Exact Test is for testing independence of row and column variables in a 2x2 table. The same ideas apply for an *I* x *J* table, though the computing time is increased. The test assumes fixed row and column totals, but as these are sufficient statistics for the row and column probabilities, which determine the null distribution, the test can be applied even if row and column totals are not fixed by design. If marginal totals were not fixed by design, an alternative is to use the "unconditional" test of Section 3.5.5, which fixes row totals only and assumes independent binomial row samples. An unconditional test may have more power than Fisher's Exact Test because the null distribution is less discrete, allowing more values for the p-value to assume. Thus, one is not forced to be as conservative as with a conditional test.

Small-sample "exact" confidence intervals for the odds ratio for a 2x2 table can be computed using Cornfield's tail method. However, if the data are highly discrete, then Agresti suggets an adjustment to the interval using the mid-p-value.

B. Confidence Intervals for Association Parameters

Wald confidence intervals, score intervals, and profile likelihood intervals are illustrated for the Aspirin and Myocardial Infarction example on p. 72 in Agresti. We briefly describe each type of interval.

1. Wald Confidence Intervals

Suppose the statistic T_n is used to estimate an unknown parameter $\theta = E(T_n)$. Wald confidence intervals are based on asymptotic normality of the standardized statistic, $\frac{(T_n-\theta)}{se(T)}$. Inverting this statistic gives the $100(1-\alpha)\%$ confidence interval on θ, $T_n \pm z_{\alpha/2}\,\sigma(T_n)$, where $z_{\alpha/2}$ is the $100(1-\alpha/2)^{th}$ quantile of the standard normal distribution, and $\sigma(T_n)$ is the standard deviation of T_n. An estimate of $\sigma(T_n)$ is given by the delta method (see Section 3.1.5 in Agresti), with sample statistics inserted where parameters are required (e.g., replace cell probabilities with sample proportions). For the odds ratio (θ), difference in proportions ($\pi_1 - \pi_2$), and relative risk (π_1/π_2) in a 2x2 table (assuming independent binomial distributions for each of the two samples), the standard errors estimated using this method are:

$$\hat{\sigma}(\log\hat{\theta}) = \left(n_{11}^{-1} + n_{12}^{-1} + n_{21}^{-1} + n_{22}^{-1}\right)^{1/2} \tag{3.1}$$

$$\hat{\sigma}(\hat{\pi}_1 - \hat{\pi}_2) = \left[\frac{\hat{\pi}_1(1-\hat{\pi}_1)}{n_1} - \frac{\hat{\pi}_2(1-\hat{\pi}_2)}{n_2}\right]^{1/2} \tag{3.2}$$

and

$$\hat{\sigma}\left(\log\hat{\pi}_1/\hat{\pi}_2\right) = \left[(1-\hat{\pi}_1)\hat{\pi}_1^{-1}n_1^{-1} + (1-\hat{\pi}_2)\hat{\pi}_2^{-1}n_2^{-1}\right]^{1/2} \tag{3.3}$$

A Wald confidence interval on either the log odds ratio or the log ratio is more stable than one constructed for the respective ratios, themselves. Take the antilog of the interval endpoints to get a confidence interval on the ratio.

We can apply these formulae to the Aspirin and Myocardial Infarction data easily using arithmetic operations in S. However, sometimes a function is useful. The function `Wald.ci` below computes asymptotic confidence intervals for the proportion difference, odds ratio, and relative risk.

To set up the data, we first do so as a data frame, then use the `design.table` function (S-PLUS) to change it to an array.

```
Drug<-c("Placebo","Aspirin")
Infarction<-c("yes","no")
table.3.1<-expand.grid(drug=Drug,infarction=Infarction)
Data<-c(28,18,656,658)
table.3.1<-cbind(table.3.1,count=Data)
(temp<-design.table(table.3.1))      # turn data frame into an array

        yes  no
Placebo  28 656
Aspirin  18 658
```

Remark: The bold line above used parentheses around an assignment statement. The effect is to make the assignment and also to print the result. (Or, more technically, apply the `print` method associated with the class of the object on the right hand side).

We apply the function `Wald.ci` below to get the confidence intervals.

```
Wald.ci<-function(Table, aff.response, alpha=.05){
    # Gives two-sided Wald CI's for odds ratio, difference in proportions and relative risk.
    # Table is a 2x2 table of counts with rows giving the treatment populations
    # aff.response is a string like "c(1,1)" giving the cell of the beneficial response and the
    #       treatment category
    # alpha is significance level

    pow<-function(x, a=-1) x^a
    z.alpha<-qnorm(1-alpha/2)

    if(is.character(aff.response))
        where<-eval(parse(text=aff.response))
    else where<-aff.response

    Next<-as.numeric(where==1) + 1

    # OR
    odds.ratio<-
    Table[where[1],where[2]]*Table[Next[1],Next[2]]/(Table[where[1],Next[2]]*Table[Next[1],where[
    2]])
    se.OR<-sqrt(sum(pow(Table)))
    ci.OR<-exp(log(odds.ratio) + c(-1,1)*z.alpha*se.OR)

    # difference of proportions
    p1<-Table[where[1],where[2]]/(n1<-Table[where[1],Next[2]] + Table[where[1],where[2]])
    p2<-Table[Next[1],where[2]]/(n2<-Table[Next[1],where[2]]+Table[Next[1],Next[2]])

    se.diff<-sqrt(p1*(1-p1)/n1 + p2*(1-p2)/n2)
    ci.diff<-(p1-p2) + c(-1,1)*z.alpha*se.diff

    # relative risk
    RR<-p1/p2
    se.RR<-sqrt((1-p1)/(p1*n1) + (1-p2)/(p2*n2))
    ci.RR<-exp(log(RR) + c(-1,1)*z.alpha*se.RR)

    list(OR=list(odds.ratio=odds.ratio, CI=ci.OR), proportion.difference=list(diff=p1-p2,
    CI=ci.diff), relative.risk=list(relative.risk=RR,CI=ci.RR))
    }
```

```
Wald.ci(temp, "c(1, 1)")      # or use Wald.ci(temp, c(1, 1))

$OR:
$OR$odds.ratio:
[1] 1.560298

$OR$CI:
[1] 0.8546703 2.8485020


$proportion.difference:
$proportion.difference$diff:
[1] 0.01430845

$proportion.difference$CI:
[1] -0.004868983  0.033485890


$relative.risk:
$relative.risk$relative.risk:
[1] 1.537362

$relative.risk$CI:
[1] 0.858614 2.752671
```

So, the death rate for the placebo group ranges from 0.85 to 2.85 times that for the aspirin group (odds ratio confidence interval). As the interval for the odds ratio covers 1.0 and somewhat below 1.0, there is still a small chance that aspirin has slight negative effects. The intervals on the relative risk and difference of proportions give similar conclusions.

Note that the relative risk estimate is very close to that of the odds ratio. This is not surprising because the sample proportion of deaths from myocardial infarction were small in both groups. The relationship between the odds ratio and relative risk (see p. 47 of Agresti) indicates that the two will be similar in this circumstance.

The same code works in R with minor changes. R does not have the `design.table` function (although with the change of one line, you can `source` it, as well as `factor.names` into R. Contact me for the one-line change). However, a glance at the function `design.table` in S-PLUS shows that the "workhorse" is the `tapply` function. So, just change the last line as follows:

```
Drug<-c("Placebo","Aspirin")
Infarction<-c("yes","no")
table.3.1<-expand.grid(drug=Drug,infarction=Infarction)
Data<-c(28,18,656,658)              # note capital D (see comment below)
table.3.1<-cbind(table.3.1,count=Data)
tapply(table.3.1$count,table.3.1[,1:2], sum) # turn data frame into an array
```

Then, use the `Wald.ci` function. Make sure that "`Data`" above is capitalized, as "`data`" is a function in the base environment of R.

2. Score Confidence Intervals

Computing a score interval on the difference of proportions ($\pi_1 - \pi_2$) is easy using some simple steps in S-PLUS. To test the null hypothesis that $\pi_1 - \pi_2 = \Delta$, we can use the test statistic $z(\Delta)$ given on p. 77 of Agresti. This statistic depends on the values of the unconstrained MLEs of π_1 and π_2, and the values of the constrained MLEs subject to the null equality, $\pi_1 - \pi_2 = \Delta$. The score confidence interval is then the set of Δ such that $|z(\Delta)| < z_{1-\alpha/2}$.

To get the constrained MLEs of π_1 and π_2 in S, we use the method of Lagrange multipliers. The Lagrangian is the log likelihood minus a constant, λ, times an expression representing the left-hand side of the constraint set equal to 0. Thus, the Lagrangian here is (up to an additive constant)

$$\begin{aligned}\mathcal{L}(\pi_1,\pi_2,\lambda) &= L(\pi_1,\pi_2\,|\,y_1,y_2,n_1,n_2) - \lambda\, c(\pi_1,\pi_2) \\ &= y_1\log(\pi_1) + (n_1-y_1)\log(1-\pi_1) + y_2\log(\pi_2) + (n_2-y_2)\log(1-\pi_2) \\ &\quad -\lambda(\pi_1-\pi_2-\Delta)\end{aligned} \tag{3.4}$$

$L(\pi_1,\pi_2\,|\,y_1,y_2,n_1,n_2)$ in (3.4) is the kernel of the log likelihood of two independent binomial counts. $c(\pi_1,\pi_2)$ is the constraint expression. To maximize $L(\pi_1,\pi_2\,|\,y_1,y_2,n_1,n_2)$ subject to $c(\pi_1,\pi_2) = 0$, we search for a point in (π_1,π_2)-space that falls along $c(\pi_1,\pi_2) = 0$ but so that the level curve of $L(\pi_1,\pi_2\,|\,y_1,y_2,n_1,n_2)$ going through that point is tangent to $c(\pi_1,\pi_2) = 0$. This point will be a local

minimizer of $L(\pi_1, \pi_2 | y_1, y_2, n_1, n_2)$ (see for example Fletcher, 1987). It can be shown that this implies that the gradients of the two functions L and c are related as follows:

$$\nabla L(\pi_1, \pi_2 | y_1, y_2, n_1, n_2) = \lambda \nabla c(\pi_1, \pi_2) \tag{3.5}$$

implying that to find a local minimizer (note that we will negate L later to find a maximizer) we must solve (3.5) and also the equation $c(\pi_1, \pi_2) = 0$. Thus, we must solve a system of three equations in three unknowns (π_1, π_2, λ), then discard the solution for the constant λ (called the Lagrange multiplier).

The function `f` below is the left-hand side of the system of equations we want to solve. This is a function that returns a three-dimensional S vector whose first two values represent the gradient of (the negative of) the Lagrangian, with respect to (π_1, π_2), with values for y_i and n_i given by the values in Table 3.1 (Agresti). The third value in the vector is $c(\pi_1, \pi_2)$.

```
f <- function(p) {
    c(-(28/p[1])+(656/(1-p[1]))-p[3], -(18/p[2])+(658/(1-p[2]))+p[3], p[1]-p[2])
        }
```

We now use the `solveNonlinear` function introduced in Chapter 1 to solve the system, where `y0` is equal to `c(0, 0, Δ)`. Note that `g` in `solveNonlinear` is a sum of squared errors between `f` and `y0`.

```
solveNonlinear <- function(f, y0, x,...){
  # solve f(x) = y0
  # x is vector of initial guesses, same length as y0
  # ... are additional arguments to nlmin (not to f)
  g <- function(x, y0, f) sum((f(x) - y0)^2)
  g$y0 <- y0   # set g's default value for y0
  g$f <- f     # set g's default value for f
  nlmin(g, x, ...)
}
```

Now, the function `score.ci` solves the equations given a value for Δ. Then, it computes `z` using the solutions for π_1 and π_2. We use the unconstrained MLE's as starting values for (π_1, π_2) and use trial and error for a starting value for λ.

```
score.ci<-function(Delta) {
    temp<-solveNonlinear(f, y0=c(0,0,Delta),
    x=c(28/(28+656),18/(18+658),7),print.level=0,max.fcal=100,max.iter=100,
    init.step=.001)
    p<-temp$x[1:2]
    z<-(p[1]-p[2]-Delta)/sqrt(p[1]*(1-p[1])/684 + p[2]*(1-p[2])/676)
}
```

Now, I find the values of Δ satisfying the inequality, $|z(\Delta)| < z_{0.975}$ for a 95% confidence interval.

```
Delta<-seq(-.2,.2,.001)
z<-sapply(Delta,score.ci)
range(Delta[abs(z)< qnorm(.975)])
```

```
[1] -0.005 0.035
```

Thus, the score interval is (-0.005, 0.035), slightly wider than the Wald interval.

The same idea can be carried out with R, using, for example, the `optim` function. Then, the following commands produce a score confidence interval. This time, the function `g` (here called `gfun`) is directly used instead of just existing within `solveNonlinear`. Also, we control the step sizes (`ndeps`) for the computation of the finite difference gradient to be much smaller than the default (see also, the R package `Bhat` with function `dfp` to possibly get around this). And, we use a set of Deltas much narrower in order to get better accuracy without needing much more computation time.

```
gfun<-function(p,y0){
    sum((c(-(28/p[1])+(656/(1-p[1]))-p[3],-(18/p[2])+(658/(1-p[2]))+p[3],p[1]-p[2])-
   y0)^2)
}

score.ci<-function(Delta) {

   temp<-optim(fn=gfun,par=c(28/(28+656),18/(18+658),7),
   method="BFGS",y0=c(0,0,Delta),
   control=list(ndeps=c(rep(.000000001,3))))

   p<-temp$par[1:2]
   z<-(p[1]-p[2]-Delta)/sqrt(p[1]*(1-p[1])/684 + p[2]*(1-p[2])/676)
}


Delta<-seq(-.05,.05,.0001)    # range only from -.05 to 0.05 to get better accuracy
z<-sapply(Delta,score.ci)
range(Delta[abs(z)< qnorm(.975)])

[1] -0.0042 0.0341
```

Thus, the confidence interval is $(-0.0042, 0.0341)$.

3. Profile Likelihood Confidence Intervals

Agresti illustrates a profile likelihood confidence for the odds ratio of a 2x2 table. He notes that the multinomial likelihood for the table can be expressed in terms of the odds ratio θ and the two marginal probabilities π_{1+} and π_{+1}. This is easily seen by writing the four equations $\theta = \pi_{11}\pi_{22}/\pi_{21}\pi_{12}$, $\pi_{+1} = \pi_{11} + \pi_{21}$, $\pi_{1+} = \pi_{11} + \pi_{12}$, and $\pi_{11} + \pi_{12} + \pi_{21} + \pi_{22} = 1$. Unfortunately, the actual expressions for $\{\pi_{11}, \pi_{12}, \pi_{21}, \pi_{22}\}$ for use in the new likelihood are rather complicated to type into an optimization program. However, if done, then the resulting likelihood can be maximized subject to setting θ equal to θ_0, and the maximized value is the profile likelihood for θ at the value θ_0.

If, as Agresti, we denote the value of the profile likelihood at θ_0 as $L(\theta_0, \hat{\pi}_{1+}(\theta_0), \hat{\pi}_{+1}(\theta_0))$, where $\hat{\pi}_{1+}(\theta_0)$ and $\hat{\pi}_{+1}(\theta_0)$ are the maximizers as a function of θ_0, and we denote as $L(\hat{\theta}, \hat{\pi}_{1+}, \hat{\pi}_{+1})$, the unconstrained log likelihood evaluated at the MLEs, then the profile-likelihood based confidence interval for θ includes all those values of θ_0 for which

$$-2\left[L(\theta_0, \hat{\pi}_{1+}(\theta_0), \hat{\pi}_{+1}(\theta_0)) - L(\hat{\theta}, \hat{\pi}_{1+}, \hat{\pi}_{+1})\right] < \chi^2_1(\alpha) \tag{3.6}$$

With a small amount of tedium, we could use `optim` or `nlminb` to maximize the constrained log likelihood for each value of θ_0 and then check condition (3.6). Luckily, the R package `Bhat` contains a function `plkhci` for profile-likelihood based confidence intervals which eliminates some of the work for us.

However, we still must type in the log likelihood as a function of θ, π_{1+}, and π_{+1}. One can find the necessary transformations very easily using a computer algebra system like Mathematica (2001)..much more readily than using S!. These are then copied almost "word" for "word" into the following function, `nlogf`:

```
library(Bhat)

# neg. log-likelihood of "new" multinomial model
nlogf <- function (p) {
    p1p<-p[1]
    pp1<-p[2]
    theta<-p[3]
    n11 <- table.3.1$count[1]     # recall table.3.1 above
    n21<-table.3.1$count[2]
    n12<-table.3.1$count[3]
    n22<-table.3.1$count[4]
# the following are the transformations from Mathematica 4.0
    p22<-(-1 + p1p + pp1 + 2*theta - (p1p + pp1)*theta - sqrt(-4*p1p*(-1 + pp1)*(-1 + theta) +
        (1 + p1p + pp1*(-1 + theta) - p1p*theta)^2))/(2*(-1 + theta))
    p11 <- (1 + p1p*(-1 + theta) + pp1*(-1 + theta) - sqrt(-4*p1p*(-1 + pp1)*(-1 + theta) +
        (1 + p1p + pp1*(-1 + theta) - p1p*theta)^2))/(2*(-1 + theta))
    p21 <- (-1 + p1p + pp1*(-1 + theta) - p1p*theta + sqrt(-4*p1p*(-1 + pp1)*(-1 + theta) +
        (1 + p1p + pp1*(-1 + theta) - p1p*theta)^2))/(2*(-1 + theta))
    p12 <- (-1 + pp1 + p1p*(-1 + theta) - pp1*theta + sqrt(-4*p1p*(-1 + pp1)*(-1 + theta) +
        (1 + p1p + pp1*(-1 + theta) - p1p*theta)^2))/(2*(-1 + theta))

    -(n11*log(p11) + n12*log(p12) + n21*log(p21) + n22*log(p22))
    }
```

Now, we must set up a list with names: `label` for the parameter names, `est` for parameter estimates (such as MLEs), and `low` and `upp` for the upper and lower bounds of the parameters.

```
x <- list(label=c("p1p","pp1","theta"),     # p1p = row marginal, pp1 = col marg
    est=c((28+656)/(684+676),(28+18)/(684+676),1.56),
    low=c(0,0,0),upp=c(1,1,100))      # needed theta < finite bound to work
```

Now, call the function with arguments, the named list, the name of the function, and finally the label of the parameter for which you want confidence bounds.

```
plkhci(x,nlogf,"theta")

neg. log. likelihood:  1142.580

 will atempt to compute both bounds (+/- direction)

trying lower bound ------------------------
starting at:    0.9352881
initial guess:  0.5029413 0.03382281 2.107252

begin Newton-Raphson search for profile lkh conf. bounds:
eps value for stop criterium: 0.001
nmax                        : 10

 CONVERGENCE:  6  iterations

chisquare value is:  3.841455
confidence bound of  theta  is  2.896089
log derivatives:     2.161155e-06 6.867801e-06
  label estimate  log deriv   log curv
1 p1p   0.502911  2.16116e-06 341.928
2 pp1   0.0336962 6.8678e-06  42.171
3 theta 2.89609   -5.75779    7.95543
```

```
trying upper bound ------------------------
starting at:    0.9856728
initial guess:  0.5029411 0.03382424 1.153263

begin Newton-Raphson search for profile lkh conf. bounds:
eps value for stop criterium: 0.001
nmax                        : 10

 CONVERGENCE:  6  iterations

chisquare value is:  3.841459
confidence bound of  theta  is  0.8617511
log derivatives:     2.122385e-06 6.523067e-06
  label estimate  log deriv   log curv
1 p1p   0.502946  2.12238e-06 339.335
2 pp1   0.0336989 6.52307e-06 41.8502
3 theta 0.861751  6.44456     10.8561

[1] 0.8617511 2.8960895
```

Thus, the profile likelihood based confidence interval for the odds ratio is: (0.862, 2.896), fairly close to the Wald interval.

C. Testing Independence in Two-way Contingency Tables

For multinomial sampling with probabilities $\{\pi_{ij}\}$ in an I x J contingency table, the null hypothesis of statistical independence of the row and column variables is $H_0 : \pi_{ij} = \pi_{i+}\pi_{j+}$ for all i and j. Pearson's chi-squared statistic can be used to test this hypothesis. The expected frequencies in an I x J contingency table under H_0 are $\mu_{ij} = n\pi_{ij} = n\pi_{i+}\pi_{j+}$, with MLEs given by $\hat{\mu}_{ij} = n_{i+}n_{j+}/n$. Then, the chi-squared statistic, X^2, is given on p. 78 of Agresti.

The score test uses Pearson's chi-squared statistics. But, as mentioned in Agresti, the likelihood ratio test uses a different "chi-squared statistic" (they are both called chi-squared statistics because they both have asymptotic chi-squared distribution under the null hypothesis of independence). The likelihood ratio chi-squared statistic is given on p. 79 of Agresti. The degrees of freedom for each asymptotic chi-squared distribution is $(I-1)(J-1)$. The limiting distributions apply when the total sample size is large and the number of cells is fixed. Agresti p.79-80 gives further details.

Agresti uses data on religious fundamentalism and degree of education to illustrate the two different chi-squared tests of independence. The data are in Table 3.2 (p.80).

First, I set up the data in S-PLUS:

```
religion.counts<-c(178,138,108,570,648,442,138,252,252)
table.3.2<-cbind(expand.grid(list(Religious.Beliefs=c("Fund", "Mod", "Lib"),
   Highest.Degree=c("<HS","HS or JH", "Bachelor or Grad"))),count=religion.counts)
(table.3.2.array<-t(design.table(table.3.2))) # t() is to arrange the table as in
   Agresti

                Fund Mod Lib
            <HS  178 138 108
       HS or JH  570 648 442
Bachelor or Grad  138 252 252
```

Now, we can use the `chisq.test` function. In S-PLUS, type

```
chisq.test(table.3.2.array)
```

```
    Pearson's chi-square test without Yates' continuity correction

data:  design.table(table.3.2)
X-square = 69.1568, df = 4, p-value = 0
```

To obtain the expected frequencies, we take advantage of the `outer` function, as well as the functions `rowSums` and `colSums`:

```
expected.freqs<-
   outer(rowSums(table.3.2.array),colSums(table.3.2.array),FUN="*")/sum(table.3.2.arra
   y)

expected.freqs

                     Fund      Mod      Lib
            <HS 137.8078 161.4497 124.7425
       HS or JH 539.5304 632.0910 488.3786
Bachelor or Grad 208.6618 244.4593 188.8789
```

In R, the data set-up uses `tapply` directly, instead of going through `design.table`, as in Section 1.B. The package `ctest` contains the function `chisq.test`, which is similar to that above in output, except that the function also returns the expected frequencies invisibly (A very nice addition. And, it happens to be computed in the same way I computed it above!). For example,

```
table.3.2.array<-tapply(table.3.2$count,table.3.2[,1:2], sum)
(res<-chisq.test(table.3.2.array))

        Pearson's Chi-squared test

data:  table.3.2.array
X-squared = 69.1568, df = 4, p-value = 3.42e-14

res$expected
                  Religious.Beliefs
Highest.Degree         Fund      Mod      Lib
  <HS              137.8078 161.4497 124.7425
  HS or JH         539.5304 632.0910 488.3786
  Bachelor or Grad 208.6618 244.4593 188.8789
```

The chi-square tests (both Pearson and LR) and expected frequencies (as well as marginal totals) can be obtained just as easily with the package `vcd`, using functions, `assoc.stats`, `expected`, and `mar.table`, respectively.

```
library(vcd)
assoc.stats(table.3.2.array)

                    X^2 df   P(> X^2)
Likelihood Ratio 69.812  4 2.4869e-14
Pearson          69.157  4 3.4195e-14

Phi-Coefficient   : 0.159
Contingency Coeff.: 0.157
Cramer's V        : 0.113

expected(table.3.2.array)

          <HS HS or JH Bachelor or Grad
Fund 137.8078 539.5304         208.6618
Mod  161.4497 632.0910         244.4593
Lib  124.7425 488.3786         188.8789
```

A nice feature of the R package version of `chisq.test` is that one has the option of computing the p-value via Monte Carlo simulation. This is very nice in cases where the expected frequencies do not meet

the rules of thumb required for the asymptotic chi-squared distribution to be appropriate, and the table is too large to use some of the exact tests offered. To request p-value computation via Monte Carlo simulation, set the argument `simulate.p.value` (or `sim`) to TRUE. The argument `B` controls the number of replicates in the simulation.

```
chisq.test(table.3.2.array, sim=T, B=2000)

        Pearson's Chi-squared test with simulated p-value (based on 2000
        replicates)

data:  table.3.2.array
X-squared = 69.1568, df = NA, p-value = < 2.2e-16
```

Here, the p-value is similarly very low, rejecting the hypothesis of independence of the two variables.

For the likelihood ratio test statistic, one can easily use the expected frequencies obtained above and the formula for the statistic on p. 79 of Agresti. For example,

```
2*sum(table.3.2.array*log(table.3.2.array/expected.freqs)) # R: Use res$expected instead
   of expected.freqs

[1] 69.81162
```

Alternatively, jumping ahead a bit, we can use the `glm` function for generalized linear models (see Chapter 4) and the Poisson distribution (which is the distribution of the counts in the table when the total number of cases is not fixed – see p. 132 in Agresti) to get the likelihood ratio chi-squared statistic. For example, in either S-PLUS or R, we get

```
fit.glm<-glm(count~Religious.Beliefs+Highest.Degree, data=table.3.2, family=poisson)
fit.glm$deviance

[1] 69.81162
```

The expected frequencies can then be obtained using the function `predict`.

```
temp<-predict(fit.glm,type="response")
matrix(temp, nc=3, byrow=T, dimnames=list(c("<HS","HS or JH", "Bachelor or
   Grad"),c("Fund", "Mod", "Lib")))

                     Fund      Mod      Lib
             <HS 137.8078 161.4497 124.7425
        HS or JH 539.5305 632.0910 488.3786
Bachelor or Grad 208.6618 244.4593 188.8790
```

D. Following Up Chi-Squared Tests

After running a chi-squared test and rejecting the hypothesis of independence, one may want to know where the association between the two variables lies. Agresti lists the use of residual analysis and partitioning the overall chi-squared statistic for subtables as methods for assessing where association may lie.

1. Standardized Pearson residuals

The (squared) Pearson residuals are actually components of the Pearson chi-squared statistic. One can extract the Pearson residuals from a `glm` object in S by using the function `residuals`, with type argument `"pearson"`. For example, using the data above,

```
resid.pear <- residuals(fit.glm, type = "pearson")
```

Note that the sum of the squared Pearson residuals equals the Pearson chi-squared statistic:

```
sum(resid.pear^2)
[1] 69.11429
```

To get the standardized residuals, just modify `resid.pear` according to the formula on p. 81 of Agresti.

```
ni<-rowSums(table.3.2.array)  # row sums
nj<-colSums(table.3.2.array)  # column sums
n<-sum(table.3.2.array)    # total sample size
resid.pear.mat<-matrix(resid.pear, nc=3, byrow=T, dimnames=list(c("<HS","HS or JH",
   "Bachelor or Grad"),c("Fund", "Mod", "Lib")))

n*resid.pear.mat/sqrt(outer(n-ni,n-nj,"*") )   # standardized Pearson residuals

                     Fund        Mod       Lib
            <HS  4.534062 -2.5520482 -1.941537
        HS or JH  2.552988  1.2859745 -3.994669
Bachelor or Grad -6.806638  0.7007539  6.250329
```

Because the standardized residuals are asymptotically standard normal, we can compare them against standard normal "critical values" such as ± 1.96 as an indication of lack of fit of the respective cell frequency to an independence model. Thus, for the religious fundamentalism example, there is strong lack of fit in three of the four corners of the table, indicating higher frequency than expected by H_0 in the cells for liberal/bachleor's and fundamentalist/<HS, and lower frequency than expected in the bachelor/fundamentalist cell.

2. Partitioning Chi-Squared

Because a chi-squared random variable with ν degrees of freedom can be partitioned into the sum of ν independent chi-squared random variables, we can also partition a chi-squared test of independence into separate independent tests, the sum of which equal the overall chi-squared statistic. As Agresti says, this partitioning may highlight where in a table association applies.

Not all partitions of an overall table yield independent component chi-squared statistics, but one that does is given on p. 83 of Agresti, and explicit rules appear on p. 84. This partitioning is illustrated with the Schizophrenia data set on p. 83, which cross-classifies a sample of psychiatrists by their school of psychiatric thought and their opinion on the origin of schizophrenia. To calculate the likelihood ratio (LR) chi-squared statistic for the test of the null hypothesis of independence of the two variables, School and Origin, we can use the `glm` function again with the Poisson family.

```
schizo.counts<-c(90,12,78,13,1,6,19,13,50)
table.3.3<-cbind(expand.grid(list(Origin=c("Biogenic", "Environmental",
   "Combination"),  School=c("Eclectic","Medical", "Psychoanalytic"))),
   count=schizo.counts)
```

The full LR test is

```
fit<-glm(count~Origin+School, family="poisson", data=table.3.3)
fit$deviance     # LR statistic

[1] 23.03619
```

which is significant at the 0.05 level, rejecting independence. Now, to test independence of the row and columns variables in Subtable 1 in Table 3.4 in Agresti p. 83, we can use the `update` function. This test compares the Eclectic and Medical schools of thought on whether the origin of schizophrenia is biogenic or environmental, given that classifications only in these last two categories are considered.

```
update(fit, subset=(School=="Eclectic" | School=="Medical") & (Origin=="Biogenic" |
   Origin=="Environmental"))$deviance
```

```
[1] 0.2941939
```

Thus, we do not reject the hypothesis of independence for Subtable 1. Subtables 2 through 4 require a bit more than just case selection via an `update` call. Subtable 2 compares the Eclectic and Medical schools on whether the origin is a combination or not. So, we use the `aggregate` function to sum columns.

First, I add a column called `select` that indicates the new categories. Then, I `aggregate` the counts in `table.3.3` by `School` and `select`, and sum these counts. This gives the new table, which is supplied as the `data` argument to `glm`. Finally, to restrict the analysis to just the Eclectic and Medical schools, I use the `subset` argument.

```
table.3.3$select<-rep(c("Bio+Env","Bio+Env","Com"),3)
table.3.3.sub2<-aggregate(table.3.3$count,by=list(School=table.3.3$School,
   select=table.3.3$select),sum)
(fit<-glm(x~select+School, family="poisson", data=table.3.3.sub2, subset=
   School=="Eclectic" | School=="Medical"))$deviance
```

```
[1] 1.358793
```

Thus, we also do not reject the null hypothesis for Subtable 2, as the LR statistic is too small (for any significance level less than about 0.24) compared to a chi-squared distribution with 1 df. So, in neither of Subtables 1 and 2 does the association between Origin of Schizophrenia and School of Thought lie.

The remaining subtable tests are calculated similarly, and the sum of their LR statistics is the LR statistic of the full table. The above functions can be used in R or S-PLUS. Pearson chi-squared tests can be conducted as before using the `chisq.test` function with the subtable data frames as arrays.

E. Two-Way Tables with Ordered Classification

The LR and Pearson chi-squared statistics ignore information about any inherent order that is present in the categories of one or more classification variables. This section deals with tests of independence of two classification factors where both factors have ordinal-level categories. The alternatives to independence that Agresti discusses in this section are a linear association between the two factors and a monotone association.

1. Linear Trend Alternative to Independence

The first test is one for detecting nonzero true correlation between two ordinal factors. It is

$$M^2 = (n-1)r^2 \tag{3.7}$$

Thus, with a larger correlation, r, or larger sample, M^2 is larger. For large samples, it is approximately chi-squared with 1 df. A small p-value indicates that there may be a strong linear component to any association.

Implementation of (3.7) in S is just a matter of computing r on chosen scores. For example, for the Job Satisfaction data in Table 2.8 of Agresti, he uses scores 1, 2, 3, and 4 for job satisfaction and scores 7.5, 20, 32.5, and 60 in thousands of dollars for income. The income scores represent approximate category midpoints. In S-PLUS or R, correlation (via the product-moment formula) is computed using the function `cor`. First, we must convert the levels of `income` and `jobsat` to numeric labels.

```
levels(table.2.8$income)<-c(7.5,20,32.5,60)
levels(table.2.8$jobsat)<-1:4
```

Then, we repeat the `income` and `jobsat` values in `table.2.8` `count` times. We can do this in at least two ways. One way is to use `apply`. The other is to just `rep` the row labels `count` times. Both methods are fairly comparable in efficiency, but method 2 requires a subsequent change of class of the columns to numeric. Then, we apply `cor`, which returns an entire matrix, from which we select the 2,1 element.

```
res<-table.2.8[,1:2][rep(1:nrow(table.2.8),table.2.8$count),]   # method 2 (see above)
res<-apply(table.2.8[,1:2],2,function(x){rep(x,table.2.8$count)})  # method 1
(cor(res)[2,1]^2)*(nrow(res)-1)
[1] 3.807461

1-pchisq(3.807461,1)  # M^2 ~~ chisq, 1 df
[1] 0.0510247
```

As the p-value is relatively low, we can conclude that an association that involves a linear component may be likely.

2. Monotone Trend Alternative to Independence

One problem with the above analysis is that the scores impose an interval scale on the two factors, which may not be appropriate because we really only have ordinal level variables. This section uses the gamma statistic (Section 2.4.4 in Agresti) to test independence against the weaker alternative of monotonicity. We can use our `Gamma.f` function from Section 2.E to compute the monotonic association between `jobsat` and `income`.

We'll modify `Gamma.f` a little to get the standard error of gamma. One modification appears below in `Gamma2.f`. This is based on problem 3.27 in Agresti. Instead of fiddling with `table.2.8` to get the matrix of counts needed for input into `Gamma2.f`, we can just reuse the call to `crosstabs` (or `xtabs`), or call it again if it wasn't saved.

There is also a slight difference in implementation of `Gamma2.f` across S-PLUS and R. Thus, I first check (using `version`) whether we are using R or S-PLUS. The `version` object is different across the two implementations. Check this for yourself.

```
Gamma2.f<-function(x, pr=0.95)
{
    # x is a matrix of counts.  You can use output of crosstabs or xtabs in R.
    # A matrix of counts can be formed from a data frame by using design.table.

    # Confidence interval calculation and output from Greg Rodd

    # Check for using S-PLUS and output is from crosstabs (needs >= S-PLUS 6.0)
    if(is.null(version$language) && inherits(x, "crosstabs")) { oldClass(x)<-NULL;
   attr(x, "marginals")<-NULL}

    n <- nrow(x)
    m <- ncol(x)
    pi.c<-pi.d<-matrix(0,nr=n,nc=m)

    row.x<-row(x)
    col.x<-col(x)

    for(i in 1:(n)){
        for(j in 1:(m)){
            pi.c[i, j]<-sum(x[row.x<i & col.x<j]) + sum(x[row.x>i & col.x>j])
            pi.d[i, j]<-sum(x[row.x<i & col.x>j]) + sum(x[row.x>i & col.x<j])
        }
    }

    C <- sum(pi.c*x)/2
    D <- sum(pi.d*x)/2
```

```
    psi<-2*(D*pi.c-C*pi.d)/(C+D)^2
    sigma2<-sum(x*psi^2)-sum(x*psi)^2

    gamma <- (C - D)/(C + D)
    pr2 <- 1 - (1 - pr)/2
    CIa <- qnorm(pr2) * sqrt(sigma2) * c(-1, 1) + gamma

    list(gamma = gamma, C = C, D = D, sigma = sqrt(sigma2), Level = paste(
        100 * pr, "%", sep = ""), CI = paste(c("[", max(CIa[1], -1),
        ", ", min(CIa[2], 1), "]"), collapse = ""))
}


temp<-crosstabs(formula = count ~ income + jobsat, data = table.2.8)
Gamma2.f(temp)

$gamma
[1] 0.2211009

$C
[1] 1331

$D
[1] 849

$sigma
[1] 0.1171628

$Level
[1] "97.5%"

$CI
[1] "[-0.00853400851071168, 0.450735843373097]"
```

Using the SE estimate of 0.117, a 95% confidence interval on γ is $(-0.01, 0.45)$.

The function `Gamma2.f` takes less than 1 CPU second on a Pentium 4 (original version) computer with 512 MB of RAM running S-PLUS 6.1, and takes even less time using R.

```
resources(res<-Gamma2.f(temp))    # S-PLUS (resources is available from Venables and
   Ripley (2000))

  CPU Elapsed % CPU Child Cache Working
 0.16    0.16   100     0     0    6274
```

F. Small Sample Tests of Independence

When the total sample size for the table is small (where "small" in practice may be defined by your computer software, as we will see shortly), exact tests and exact confidence intervals are preferred to their large-sample "equivalents" for obvious reasons. In this section I describe how to use S for the exact tests discussed in Section 3.5 in Agresti for testing independence of row and column variables.

1. Fisher's Exact Test

For 2x2 tables, given the marginal totals, the entire table is determined by one cell count (say the 1,1 cell). Under independence, this cell count has the hypergeometric distribution given in (3.16) in Agresti. (In general, it has a noncentral hypergeometric distribution with noncentrality parameter the odds ratio). For a 2x2 table, Fisher's Exact Test is a test of the null hypothesis that the odds ratio corresponding to the table is 1. A one-sided p-value of the test is the sum of all hypergeometric probabilities that are more

consistent with the alternative hypothesis than the one corresponding to the observed table, plus the probability corresponding to the observed table. One possible two-sided p-value is double this result, provided the answer does not exceed 1. Other possibilities are given in Agresti p. 93. Some of these can be implemented using `dhyper` (see below).

To demonstrate Fisher's Exact Test, Agresti uses the data from Table 3.8 (p. 92). The function `fisher.test` in S-PLUS gives a two-sided test; the version in R gives either a one- or two-sided test. The function can be used for *I* x *J* tables as well, with *I*, *J* $\leq$ 10, but the total sample size in the table cannot exceed 200. (Interesting, the "Tea Tasting" example is given in the help page for `fisher.test` in R). In S-PLUS we get

```
(test<-fisher.test(matrix(c(3,1,1,3),byrow=T,ncol=2)))

   Fisher's exact test

data:  matrix(c(3, 1, 1, 3), byrow = T, ncol = 2)
p-value = 0.4857
alternative hypothesis: two.sided
```

To get the one-sided p-value, type:

```
test$p.value/2
[1] 0.2428572
```

Note that the one-sided p-value can be obtained using `dhyper`, as might be expected.

```
sum(dhyper(q=c(3,4),m=4,n=4,k=4))
[1] 0.2428571
```

In R, for 2x2 tables, we can specify the alternative to be "greater". Plus, we get a confidence interval result.

```
library(ctest)
(fisher.test(matrix(c(3,1,1,3),byrow=T,ncol=2), alternative="greater"))

        Fisher's Exact Test for Count Data

data:  matrix(c(3, 1, 1, 3), byrow = T, ncol = 2)
p-value = 0.2429
alternative hypothesis: true odds ratio is greater than 1
95 percent confidence interval:
 0.3135693       Inf
sample estimates:
odds ratio
  6.408309
```

The R version allows for a null value of the odds ratio to be something other than 1 (using argument `or`).

Mid-p-values can be calculated using `dhyper`.

2. Unconditional Test of independence

If only row totals are fixed by design, a better alternative might be to use the unconditional small-sample test of independence described in Section 3.5.5 in Agresti. If we assume that rows represent independent binomial samples, then the hypothesis of independence (or homogeneity of proportions) is that the column (or response) probabilities are equal across rows ($H_0 : \pi_1 = \pi_2 = \pi$). Using a test statistic such as Pearson's chi-squared, the p-value is the supremum over $0 \leq \pi \leq 1$ of the probability that Pearson's chi-squared meets or exceeds the observed value. For the (3, 0 / 0, 3) table given on p.

95 of Agresti, $X^2 = 6$ and we need the supremum of $P(X^2 \geq 6) = 2\pi^3(1-\pi)^3$. In S, we can use general optimization functions for this. The function `optim` exists for both R and S-PLUS implementations (in S-PLUS it resides in the MASS library).

```
library(MASS) # S-PLUS only
(res<-optim(par=.25, fn=function(pi){ log(2) + 3*log(pi) + 3*log(1-pi)},
   method="L-BFGS-B", lower=.00001, upper=.9999, control=list(fnscale=-1)))

$par:
[1] 0.5

$value:
[1] -3.465736


. . .(snip)

exp(res$value)   # p-value
[1] 0.03125
```

Different ranges of π can be used by changing the `lower` and `upper` arguments.

3. Fisher's Exact Test for *I* x *J* Tables

For product multinomial sampling or ordinary multinomial sampling, conditioning on both the row and column totals, under independence, yields the multiple hypergeometric distribution for the cell counts. As mentioned in Agresti, the p-value of the test of independence is the probability of the set of tables with the given margins that are no more likely to occur than the table observed. Sometimes the tables are ordered using a statistic that describes distance from independence (such as Pearson's chi-squared). For a table with ordinal categories, the statistic might measure positive association (such as gamma).

Table 3.9 in Agresti cross-classifies level of smoking and myocardial infarction for a sample of women in a case-control study. We can use an exact conditional test of independence for ordered categories that uses the gamma statistic and calculates for the p-value, the probability of observing a gamma as large as that corresponding to Table 3.9, under independence. This will involve the multiple hypergeometric probability mass function given in equation (3.19) of Agresti.

We can easily calculate the p-value using the functions `prod` and `factorial`, but the function `fisher.test` can also be used, as it now handles *I* x *J* tables.

```
(table.3.9<-matrix(c(25,25,12,0,1,3),byrow=T,ncol=3))

     [,1] [,2] [,3]
[1,]   25   25   12
[2,]    0    1    3

Gamma.f(table.3.9)  # observed gamma

$gamma:
[1] 0.8716578

$C:
[1] 175

$D:
[1] 12
```

To get the p-value, we can use the functions `factorial` and `prod`. For R, we must use the function `fact` instead of `factorial`. `fact` is available in library `combinat`.

```
num<-prod(factorial(rep(1,2)%*%table.3.9))*prod(factorial(rep(1,3)%*%t(table.3.9)))
den<-factorial(sum(table.3.9))*prod(factorial(table.3.9))
term1<-num/den

temp<-matrix(c(25,26,11,0,0,4), byrow=T, ncol=3) # only other table with C-D as least
   as large

num<-prod(factorial(rep(1,2)%*%temp))*prod(factorial(rep(1,3)%*%t(temp)))
den<-factorial(sum(temp))*prod(factorial(temp))
term2<-num/den

term1+term2   # sum the two probabilities
[1] 0.01830808
```

Otherwise, we can use `fisher.test`:

```
fisher.test(table.3.9)$p.value/2
[1] 0.01704545
```

G. Small-Sample Confidence Intervals For 2x2 Tables

Conditional on the marginal totals, the distribution of the (1, 1) cell in a 2x2 table is noncentral hypergeometric, with non-centrality parameter the odds ratio, θ. The distribution is given on p. 99 of Agresti. A confidence interval for the odds ratio results from inverting the test of $H_0 : \theta = \theta_0$, given the observed cell counts. Cornfield's (1956) tail method for constructing a confidence interval on θ is given on p. 99 on Agresti. Briefly, the lower endpoint of a $100(1-\alpha)$% confidence interval is θ_0 for which the p-value equals $\alpha/2$ in testing against an alternative hypothesis that $\theta > \theta_0$ (the value of θ_0 that would cause us to "just" accept H_0 with the observed cell counts). The upper endpoint is the value of θ_0 for which the p-value equals $\alpha/2$ in testing against an alternative hypothesis that $\theta < \theta_0$. Because of the discreteness of the p-value, the interval is the set of θ_0 for which both one-sided p-values $\geq \alpha/2$.

As mentioned previously, the `fisher.test` function in R outputs a confidence interval on the odds ratio for a 2x2 table. For the tea tasting data, it gives

```
library(ctest)
res<-fisher.test(matrix(c(3,1,1,3),byrow=T,ncol=2), alternative="two.sided", or=1)
res$conf.int

[1]   0.2117329 621.9337505

attr(,"conf.level")
[1] 0.95
```

and it gives the estimate

```
res$estimate

odds ratio
  6.408309
```

The estimate of the odds ratio is the *conditional* MLE of θ discussed by Agresti, that is, the value of θ that maximizes the likelihood of the cell counts, conditional on the marginal totals.

Liao and Rosen (2001) give an R function for probability calculations from a non-central hypergeometric distribution. Their function can be used to obtain the confidence interval above. The syntax (using Agresti's notation) is $hypergeometric(n_{1+}, n_{+1}, n, \theta)$. This returns a set of functions such as the cumulative distribution function (CDF), the probability mass function, and moment functions. The confidence interval can be found by trial and error via the CDF, changing the odds ratio each time. Or, one can use `optim`. We can determine both endpoints with the same function by summing the squared differences of the two p-values from 0.025.

```
f<-function(x, alpha,t0){
    resl<-hypergeometric(4,4,8,x[1])
    resu<-hypergeometric(4,4,8,x[2])
    sum(c(1-resl$p(t0-1) - alpha/2, resu$p(t0)-alpha/2)^2)
}

optim(par=c(.22, 622), fn=f, method="BFGS", alpha=.05, t0=3, control=
list(parscale=c(1,100)))

$par
[1]   0.2117342 626.2385687

$value
[1] 1.345465e-13

$counts
function gradient
      91       80

$convergence
[1] 0
```

So, the two endpoints are (0.211, 626.24), which we check using `hypergeometric`.

```
res<-hypergeometric(4,4,8,626.24)    # Upper endpoint
res$p(3)  # p is the CDF returned by hypergeometric. we evaluate it at t=3
[1] 0.02500014

res<-hypergeometric(4,4,8,.212)      # lower endpoint
1-res$p(2)
[1] 0.02505818
```

As both probabilities just exceed 0.025, we take the corresponding odds ratios as the endpoints. Note the slight difference in the result compared with `fisher.test` above.

Using the non-central hypergeometric distribution function for the mid-p-value-adjusted confidence interval is just as easy.

```
f<-function(x, alpha,t0){
    resl<-hypergeometric(4,4,8,x[1])
    resu<-hypergeometric(4,4,8,x[2])
    sum(c(1-resl$p(t0-1) -.5*resl$d(t0) - alpha/2, resu$p(t0-1) + .5*resu$d(t0) -
alpha/2)^2)
}

optim(par=c(.22, 622), fn=f, method="BFGS", alpha=.05, t0=3, control=
list(parscale=c(1,100)))

$par
[1]   0.3100547 308.5567363

$value
[1] 6.545662e-16
```

```
$counts
function gradient
      58       46

$convergence
[1] 0

res<-hypergeometric(4,4,8,308.5567363)
res$p(2) + .5*res$d(3)
[1] 0.02500000

res<-hypergeometric(4,4,8,0.3100547)
1-res$p(2) - .5*res$d(3)
[1] 0.02499998
```

Thus, the confidence interval is approximately (0.31, 309).

I have neglected to mention that the `hypergeometric` function, as it is, won't work in S-PLUS because it uses R scoping rules that S-PLUS doesn't follow. The functions returned by `hypergeometric` access variables that belong to the environment in which the functions were created (i.e., the `hypergeometric` environment). Thus, all the returned functions can use any variables created within `hypergeometric`. The same won't work in S-PLUS because a function created in S-PLUS only has access to the objects created in the evaluation frame (the environment created with the function) or that exist in the parent frame (the environment in which the function was invoked) or on the search path. It is possible to modify the function so that it will work in S-PLUS. A simple way to do so is to pass as arguments all variables that are needed. Another way is to use a function that can make a closure (see the MC function in the Appendix of Gentleman and Ihaka (2000)). A modified version of the `hypergeometric` function is given below, that uses MC, as well as `substitute()`:

```
hypergeometric.SPLUS <- function(n1, m1, N, psi)
 {
    n2 <- N - n1;
    if(n1<0 | n2<0 | m1<0 | m1>N | psi<=0) stop("wrong argument in hypergeometric");

    ll <- max(0, m1-n2);
    uu <- min(n1, m1);

    prob <- array( 1, uu-ll+1 );

    shift <- 1-ll;

    mode.compute <- substitute(function()
  {
     a <- psi - 1;
     b <- -( (m1+n1+2)*psi + n2-m1 ) ;
     c <- psi*(n1+1)*(m1+1);
     q <- b + sign(b)*sqrt(b*b-4*a*c);
     q <- -q/2;

     mode <- trunc(c/q);
     if(uu>=mode && mode>=ll) return(mode)
     else return( trunc(q/a) );
  }   , list(psi=psi, m1=m1, n1=n1,ll=ll, uu=uu, n2=n2))

    mode <- mode.compute();

    r.function <- substitute(function(i) (n1-i+1)*(m1-i+1)/i/(n2-m1+i)*psi, list(psi=psi,
  m1=m1, n1=n1,ll=ll, uu=uu, n2=n2));

  if(mode<uu) #note the shift of location
  {
     r1 <- r.function( (mode+1):uu );
     prob[ (mode+1 + shift):(uu + shift) ] <- cumprod(r1);
    }
```

```
  if(mode>ll)
  {
     r1 <- 1/r.function( mode:(ll+1) );
     prob[ (mode-1 + shift):(ll + shift) ] <- cumprod(r1);
  }

  prob <- prob/sum(prob);

  mean <- function() sum( prob[(ll:uu)+shift]*(ll:uu) );

  var <-  function() sum( prob[(ll:uu)+shift]*(ll:uu)^2 ) - mean()^2;

  d <- substitute(function(x) return(prob[x + shift]), list(prob=prob, shift=shift, ll=ll,
uu=uu));

  p <- substitute(function(x, lower.tail=TRUE)
  {
      if(lower.tail) return( sum(prob[ll:(x+shift)]) )
      else return( sum( prob[(x+shift):uu] ) );
  },list(prob=prob, shift=shift, ll=ll, uu=uu))


 sample.low.to.high <- function(lower.end, ran)
 {
 for(i in lower.end:uu)
   {
      if(ran <= prob[i+shift]) return(i);
      ran <- ran - prob[i+shift];
   }
 }

sample.high.to.low <- function(upper.end, ran)
{
   for(i in upper.end:ll)
     {
       if(ran <= prob[i+shift]) return(i);
       ran <- ran - prob[i+shift];
     }
  }


  r <- function()
  {
      ran <- runif(1);

      if(mode==ll) return( sample.low.to.high(ll, ran) );
      if(mode==uu) return( sample.high.to.low(uu, ran) );

      if(ran < prob[mode+shift]) return(mode);
      ran <- ran - prob[mode+shift];

      lower <- mode - 1;
      upper <- mode + 1;

      repeat
      {
        if(prob[upper + shift] >= prob[lower + shift])
        {
          if(ran < prob[upper+shift]) return(upper);
          ran <- ran - prob[upper+shift];
          if(upper == uu) return( sample.high.to.low(lower, ran) );
          upper <- upper + 1;
        }

        else
        {
          if(ran < prob[lower+shift]) return(lower);
          ran <- ran - prob[lower+shift];
          if(lower == ll) return( sample.low.to.high(upper, ran) );
          lower <- lower - 1;
        }
      }
```

```
    }

      MC(mean, list(prob=prob, shift=shift, ll=ll, uu=uu))
      MC(var, list(prob=prob, shift=shift, ll=ll, uu=uu))
      MC(sample.low.to.high, list(prob=prob, shift=shift, ll=ll, uu=uu)      )
      MC(sample.high.to.low, list(prob=prob, shift=shift, ll=ll, uu=uu)    )
      MC(r, list(prob=prob, shift=shift, ll=ll, uu=uu)      )

    return(mean, var, d, p, r);
  }
```

Thus, the small-sample confidence interval on the odds ratio, and its adjustment using the mid-p-value can be obtained using the same commands as above:

```
f<-function(x, alpha,t0){
    resl<-hypergeometric.SPLUS(4,4,8,x[1])
    resu<-hypergeometric.SPLUS(4,4,8,x[2])
    sum(c(1-resl$p(t0-1) - alpha/2, resu$p(t0)-alpha/2)^2)
}

library(MASS)
optim(par=c(.22, 622), fn=f, method="BFGS", alpha=.05, t0=3,
   control=list(parscale=c(1,100), trace=1))

$par:
[1]   0.2117342 626.2385697

$value:
[1] 1.345311e-013

$counts:
 function gradient
       92       80

$convergence:
[1] 0
```

And for the mid-p-value:

```
f<-function(x, alpha,t0){
    resl<-hypergeometric.SPLUS(4,4,8,x[1])
    resu<-hypergeometric.SPLUS(4,4,8,x[2])
    sum(c(1-resl$p(t0-1) -.5*resl$d(t0) - alpha/2, resu$p(t0-1) + .5*resu$d(t0) -
   alpha/2)^2)
}

optim(par=c(.22, 622), fn=f, method="BFGS", alpha=.05, t0=3,
   control=list(parscale=c(1,100), trace=1))

$par:
[1]   0.3100547 308.5567363

$value:
[1] 6.546435e-016

$counts:
 function gradient
       57       46

$convergence:
[1] 0

$message:
NULL
```

The small-sample confidence interval on the difference of proportions, mentioned in Section 3.6.4 of Agresti, can be computed using the methodology from Section 3.F.2 of this manual.

Chapter 4: Generalized Linear Models

A. Summary of Chapter 4, Agresti

Chapter 4 in Agresti deals with generalized linear models (GLIMs). GLIMs extend the general linear model by allowing nonnormal response distributions and allowing a nonlinear mean function. There are three components of a GLIM, which are detailed on p. 116-117 of Agresti. Briefly, the *random component* consists of independent observations from a distribution in the natural exponential family. The pdf for this family is given in equation (4.1) of Agresti. Special discrete random variable cases are the Poisson and binomial distributions. The *systematic component* relates a vector, $\boldsymbol{\eta} = (\eta_1, ...\eta_n)^T$, to a set of explanatory variables through a linear model: $\boldsymbol{\eta} = \mathbf{X}\boldsymbol{\beta}$. $\boldsymbol{\eta}$ is called the "linear predictor". The *link function* links the random and systematic components. It describes the relationship between the mean of the response, μ_i, and the linear predictor, $\eta_i = g(\mu_i)$. When we specify a link function, g, we are saying that the systematic effects are additive on the scale given by the link function.

A fourth component is sometimes specified explicitly. This is the variance function, which is a function relating the variance to the mean (see Section 4.4 in Agresti). It is proportional to the variance of the response distribution, with proportionality constant the inverse of a parameter called the *dispersion parameter*. (If we use a particular random component, we automatically accept its variance function. However, there are methods where we can use a particular variance function that we believe describes the random phenomenon, but then refrain from "choosing" a distribution for the random component. These methods use what are called "quasi-likelihood functions".)

Typical cases of GLIMs are the binomial logit model (Bernoulli response with log odds link function) and the Poisson loglinear model (Poisson response with log link). Other cases are given in Table 4.1 in Agresti. For binomial and Poisson models, the dispersion parameter is fixed at 1.

GLIMs are fit by solving the set of likelihood equations. This leads to maximum likelihood estimates of the coefficients of the linear predictor. As the likelihood equations are usually nonlinear in the coefficients $\boldsymbol{\beta}$ of the linear predictor, the solutions are found iteratively. Iterative Reweighted Least Squares (IRLS) is the iterative method commonly used to fit GLIMs (it is used in S-PLUS). It uses Fisher scoring, which is based on the Newton-Raphson method, which achieves second-order convergence of the estimates. The difference between the two algorithms lies in the use of the observed information matrix for Newton-Raphson and the expected information matrix for Fisher scoring (see p. 145-146, Agresti). For canonical link models, these are the same. Fisher scoring will produce the estimated asymptotic covariance matrix as a by-product, but it need not have second-order convergence. Plus, the observed information matrix may be easier to calculate for complex models. The name IRLS comes from the iterative use of weighted least squares estimation, where the weights and responses (linearized forms of the link function evaluated at the observed data) change at each iteration. It is explained on p. 147 of Agresti that IRLS and Fisher scoring are the same thing.

Model deviance is the LR statistic for testing the null hypothesis that the model holds against the general alternative of a saturated model. It is twice the difference between the saturated log likelihood and the log likelihood maximized under the restrictions of the model being tested. In certain cases, this quantity has an asymptotic chi-squared distribution. If the dispersion parameter is not fixed at 1, then twice the difference between the saturated log likelihood and the restricted log likelihood is equal to the deviance scaled by the dispersion parameter (hence called the *scaled deviance*). Model deviance for a two-way contingency table is equivalent to the likelihood ratio chi-squared statistic. The deviance has an approximate chi-squared distribution for large Poisson expected values and large binomial sample sizes per covariate combination. Thus, the model deviance for Bernoulli data (0/1, instead of counts out of a total) is not asymptotically chi-squared.

One can compare two nested models (i.e., one model is a subset of the other) using the difference between their deviance values. The deviance for the larger model (more parameters to estimate) will be smaller. The comparison proceeds by first assuming that the larger model holds and testing to see if the smaller model is not statistically significantly worse in deviance. The difference in deviance is then a LRT and has an asymptotic chi-squared null distribution. For binomial or Bernoulli

data, the *difference* in deviance is the same, unlike their respective model deviances. Thus, the chi-squared approximation holds for both. In general, the use of the chi-squared approximation is much more reliable for differences of deviances than model deviances themselves (see also, McCullagh and Nelder, 1989). Model comparison is examined in detail in later chapters. Standardized Pearson and deviance residuals are additional ways to assess the fit of a model.

When we want to fit a GLIM such as the Poisson loglinear model to a data set, but the observed variance of the data is greater than that allowed by the Poisson model, we may have a case for fitting an *overdispersed* version of the model. If overdispersion is the result of subject heterogeneity, where subjects within each covariate combination still differ greatly (perhaps because we didn't measure enough covariates), then a random effects version of the model (e.g., random effects Poisson regression, random effects logistic regression) may be appropriate. Another alternative is to fit a model with a random component that allows for a greater variance than the ordinary Poisson or binomial. Some examples are the negative binomial (for random count data) and the beta-binomial (for counts out of a total).

A third alternative is to use quasi-likelihood estimation. In quasi-likelihood estimation, we take advantage of the fact that the likelihood equations for GLIMS depend on the assumed response distribution only through its mean and variance (which may be a function of the mean). Distributions in the natural exponential family are characterized by the relationship between the mean and the variance. Quasi-likelihood estimation is determined by this relationship. Thus, if we wanted to assume that the variance of a random count was some specified function of its mean, but not equal to it, we could use quasi-likelihood estimation to estimate the coefficients of the linear predictor.

Generalized Additive Models (GAMs) further generalize GLIMs by replacing the linear predictor with smooth functions of the predictors (one for each predictor). A commonly used smooth function is the cubic spline. Each smooth function is assigned a degrees of freedom, which determines how rough the function will be. The GLIM is the special case of each smooth function being a linear function. GAMs are fit via penalized maximum likelihood estimation in S-PLUS (Chambers and Hastie, 1992). GAMs have an advantage over procedures like *lowess* because they recognize the form of the response variable and only give predictions within its bounds, which is not true of *lowess* (although, one can use a *lowess* function in a GAM). In general, the advantage of GAMs over GLIMS is that the form of the predictors does not need to satisfy a particular functional relationship, like linear, logarithmic, etc. Finally, GAMs may be used in an exploratory sense by determining a parametric function to use for a predictor based on its fitted smooth function.

B. Generalized Linear Models for Binary Data

Suppose the response is binary, taking one of two possible outcomes. Then, three special cases of the GLIM use an identity link (linear probability model), a logit link (logistic regression model), and an inverse normal CDF link (probit regression model). I briefly remind you what these models are, then fit some data to them using functions in S-PLUS and R.

For a binary response, the regression model

$$\pi(\mathbf{x}) = \alpha + \beta\mathbf{x}$$

is called a linear probability model because it purports a linear relationship between the probability of positive response and the explanatory variables. Although it has a simple interpretation, the linear probability model has a structural defect in that the predicted probability of positive response can exceed 1 or be less than 0, due to the link function being the identity.

The regression model

$$\pi(\mathbf{x}) = \frac{\exp(\alpha + \beta\mathbf{x})}{1 + \exp(\alpha + \beta\mathbf{x})}$$

is called a logistic regression model. This model corresponds to a binomial GLIM with log odds link (i.e., $g(\pi(\mathbf{x})) = \log(\pi(\mathbf{x})/(1-\pi(\mathbf{x})))$).

The regression model

$$\pi(\mathbf{x}) = \Phi(\alpha + \beta\mathbf{x})$$

for a normal(0, 1) cdf, Φ, is called a probit regression model. It is a binomial GLIM with link function $\Phi^{-1}(\pi(\mathbf{x}))$.

The decision to use each of these link functions can depend on the expected rate of increase of $\pi(\mathbf{x})$ as a function of $\mathbf{x}$ or can depend on a comparison of appropriate goodness-of-fit measures (note that otherwise identical models with different link functions are not nested models).

To illustrate a logit, probit, and linear probability model, Agresti uses Table 4.2 (Snoring and heart disease, p. 121). The response variable is whether the subject had heart disease. The explanatory variable is the subject's spouse's report of the level of snoring (never, occasionally, nearly every night, every night). These levels are changed into scores (0, 2, 4, 5). The levels of snoring are treated as independent binomial samples.

To fit the three models using iterative reweighted least squares (IRLS), we can use the function `glm`, with `family="binomial"` (for the logit and probit links, at least). However, `glm` doesn't have a Newton-Raphson method or any other type of optimization method. Thus, for more general maximum likelihood estimation, we might use the function `optim` (both S-PLUS and R). Conveniently, Venables and Ripley (2002) wrote a function for maximum likelihood estimation for a binomial logistic regression. We can make minor changes to fit the linear probability model and the probit regression model.

To set up the data, type,

```
n<-c(1379, 638, 213, 254)
snoring<-rep(c(0,2,4,5),n)
y<-rep(rep(c(1,0),4),c(24,1355,35,603,21,192,30,224))
```

To fit a GLIM using maximum likelihood estimation, we use the following function slightly modified from Venables and Ripley (2002, p. 445). The default optimization method is Nelder-Mead, which is useful in many cases. Set `method="BFGS"` for a quasi-Newton method.

```
logitreg <- function(x, y, wt = rep(1, length(y)),
                intercept = T, start = rep(0, p), ...)
{
    if(!exists("optim")) library(MASS)
  fmin <- function(beta, X, y, w) {
      p <- plogis(X %*% beta)
      -sum(2 * w * ifelse(y, log(p), log(1-p)))
  }
  gmin <- function(beta, X, y, w) {
      eta <- X %*% beta; p <- plogis(eta)
      t(-2 * (w *dlogis(eta) * ifelse(y, 1/p, -1/(1-p))))%*% X
  }
  if(is.null(dim(x))) dim(x) <- c(length(x), 1)
  dn <- dimnames(x)[[2]]
  if(!length(dn)) dn <- paste("Var", 1:ncol(x), sep="")
  p <- ncol(x) + intercept
  if(intercept) {x <- cbind(1, x); dn <- c("(Intercept)", dn)}
  if(is.factor(y)) y <- (unclass(y) != 1)
  fit <- optim(start, fmin, gmin, X = x, y = y, w = wt, ...)
  names(fit$par) <- dn
  cat("\nCoefficients:\n"); print(fit$par)
```

```
  cat("\nResidual Deviance:", format(fit$value), "\n")
  cat("\nConvergence message:", fit$convergence, "\n")
  invisible(fit)
}
```

The function is written for binomial logistic regression, but is easily modified for probit regression and linear probability regression.

Thus, to fit a linear probability model, we change the functions `fmin` and `gmin` within `logitreg` to read

```
  fmin <- function(beta, X, y, w) {
      p <- X %*% beta
      -sum(2 * w * ifelse(y, log(p), log(1-p)))
  }
  gmin <- function(beta, X, y, w) {
      p <- X %*% beta;
      t(-2 * (w * ifelse(y, 1/p, -1/(1-p))))%*% X
   }
```

These are the objective function and gradient function, respectively. For probit regression, we change `fmin` and `gmin` to read

```
  fmin <- function(beta, X, y, w) {
      p <- pnorm(X %*% beta)
      -sum(2 * w * ifelse(y, log(p), log(1-p)))
  }
  gmin <- function(beta, X, y, w) {
      eta <- X %*% beta; p <- pnorm(eta)
      t(-2 * (w *dnorm(eta) * ifelse(y, 1/p, -1/(1-p))))%*% X
   }
```

So, the respective fits are obtained with:

```
(logit.fit<-logitreg(x=snoring, y=y, hessian=T, method="BFGS"))

Coefficients:
 (Intercept)     Var1
   -3.866245 0.397335

Residual Deviance: 837.7316

Convergence message: 0


(lpm.fit<-lpmreg(x=snoring, y=y, start=c(.05,.05), hessian=T, method="BFGS"))

Coefficients:
  1: NAs generated in: log(x)
  2: NAs generated in: log(x)
  3: NAs generated in: log(x)
 (Intercept)       Var1
  0.01724645 0.01977784

Residual Deviance: 834.9919

Convergence message: 0


(probit.fit<-probitreg(x=snoring, y=y, start=c(-3.87,.40)))

Coefficients:
 (Intercept)      Var1
    -2.06055 0.1877702
```

```
Residual Deviance: 836.7943

Convergence message: 0
```

The warnings given with the linear probability model are somewhat expected, as the probability can be less than 0 or greater than 1. Also, note that we cannot use the default starting values.

Approximate standard errors for the two parameter estimates can be obtained using the inverse of the *observed* information matrix.

```
sqrt(diag(solve(logit.fit$hessian)))
[1] 0.11753115 0.03536285

sqrt(diag(solve(lpm.fit$hessian)))
[1] 0.002426329 0.001976457

sqrt(diag(solve(probit.fit$hessian)))
[1] 0.04981505 0.01670894
```

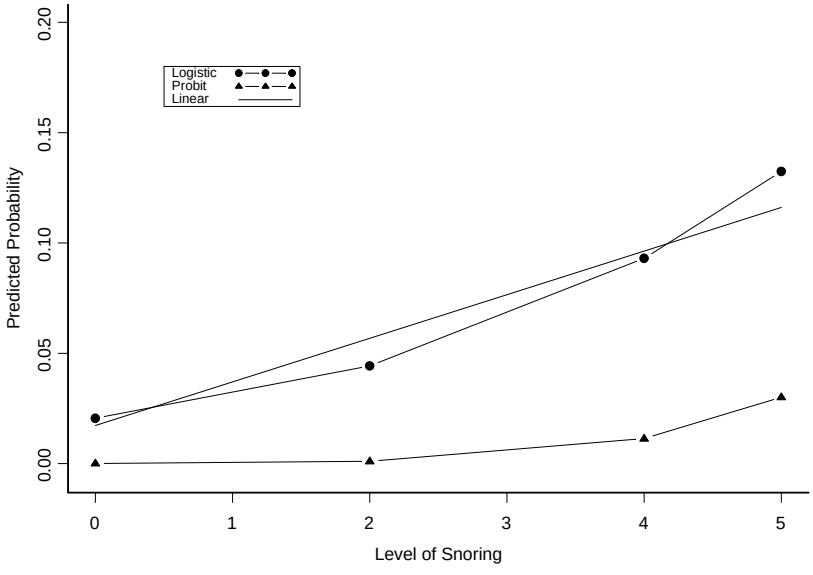
We can obtain fitted probabilities for all three link functions.

```
eta<-cbind(1,snoring)%*%logit.fit$par
logit.probs<-(exp(eta)/(1+exp(eta)))

eta<-cbind(1,snoring)%*%lpm.fit$par
lpm.probs<-(eta)

eta<-cbind(1,snoring)%*%logit.fit$par
probit.probs<-(pnorm(eta))

res<-cbind(logit=unique(logit.probs), lpm=unique(lpm.probs),
   probit=unique(probit.probs))
dimnames(res)[[1]]<-unique(snoring)
# R: dimnames(res)<-list(unique(snoring),NULL)
res
```

```
       logit        lpm        probit
0 0.02050626 0.01724271 0.00005524827
2 0.04428670 0.05683766 0.00106395563
4 0.09302543 0.09643260 0.01138590189
5 0.13239167 0.11623008 0.03005570088
```

To plot the predicted probabilities, use the following commands:

```
snoring.plot<-unique(snoring)
plot(snoring,logit.probs,type="n",xlim=c(0,5),ylim=c(-.005,.20),xlab="Level of
   Snoring",
   ylab="Predicted Probability", bty="L")
lines(snoring.plot,unique(logit.probs),type="b",pch=16)
lines(snoring.plot,unique(probit.probs),type="b",pch=17)
lines(snoring.plot,unique(lpm.probs),type="l",lty=1)
key(x=.5,y=.18,text=list(c("Logistic","Probit","Linear")),
lines=list(type=c("b","b","l")),lty=c(1,1,1),pch=c(16,17,1),divide=3,border=T)

# R: legend(x=.05,y=.18,legend=c("Logistic","Probit","Linear"), lty=c(1,1,1),
   pch=c(16,17,-1), cex=.85, text.width=1, adj=-.5)
```



Estimation with IRLS

If you were wondering what estimates and standard errors IRLS would give, you can fit all three of the above models using the `glm` function, which is described in more detail in the next section. For example, to fit the binomial with logit link using IRLS, type

```
snoring<-c(0,2,4,5)

logit.irls<-glm(cbind(yes=c(24,35,21,30), no=c(1355,603,192,224))~snoring,
   family=binomial(link=logit))

summary(logit.irls)$coefficients

Coefficients:
                Value Std. Error    t value
(Intercept) -3.8662481 0.16620356 -23.262125
    snoring  0.3973366 0.05000865   7.945358
```

The difference in the standard errors as compared with the ML fit above (for the logit link) has to do with the fact that the hessian is numerically differentiated in the first fit.

Similarly, the probit regression model is fit using

```
probit.irls<-glm(cbind(yes=c(24,35,21,30), no=c(1355,603,192,224))~snoring,
   family=binomial(link=probit))

summary(probit.irls)$coefficients

                Value Std. Error    t value
(Intercept) -2.0605515 0.07016609 -29.366769
    snoring  0.1877704 0.02348045   7.996883
```

Fitting the linear probability model using IRLS and `glm` requires using the `quasi` family with `identity` link (and `variance` function implicitly set to constant). Specifying the quasi family uses a quasi-likelihood instead of a binomial likelihood (for example). The score equations for both the `binomial(link=identity)` and `quasi(link=identity)` estimations are the same. Thus, the estimates will be the same, without having to specify a distribution for the response. Quasi-likelihood is discussed in Section 4.7 of Agresti.

Notice also that, in the formula, we use the proportions for each category as the response, plus we add the `weights` argument, which is the total number in each category.

```
prop<-c(24/1379, 35/638, 21/213, 30/254)
lpm.irls<-glm(prop~snoring, weights=c(1379,638,213,254),family=quasi(link=identity))

summary(lpm.irls)$coefficients
                Value   Std. Error  t value
(Intercept) 0.01687233 0.0011341069 14.87719
    snoring 0.02003800 0.0005094491 39.33269
```

All of the IRLS estimates are similar to the MLEs.

C. Generalized Linear Models for Count Data

The Poisson distribution is frequently used for modeling responses that are counts. A Poisson loglinear GLIM assumes a Poisson distribution for the response and the log function for the link function. So, the linear predictor is related to the mean as

$$\mu = \exp(\mathbf{X}\boldsymbol{\beta}) \tag{4.1}$$

Thus, explanatory variables are modeled to have multiplicative impacts on the mean response.

Agresti uses the Horseshoe Crab data to fit a Poisson generalized linear model. Each female crab in the data set had a male crab resident in her nest. The response variable measured was the number of additional males (satellites) residing nearby each female. Explanatory variables were the female crab's color, spine condition, weight and carapace width. Width is the only explanatory variable used to fit the Poisson model in this section. This data set is available on Agresti's text book web site. I copied it into a text file, which I called `crab.ssc`.

First, however, we can reproduce Figures 4.3 and 4.4 on pages 128 and 129 of Agresti. Figure 4.3 plots the number of satellites by carapace width, with numbered symbols indicating the number of observations at each point. We first read in the data set. Then, to plot the numbered symbols, we first `aggregate` the response and explanatory variables by unique pairs, then sum the number of observations with those pairs. These sums are what appear as the symbols. I explain each step below.

First I read in the data

```
table.4.3<-read.table("crab.ssc", col.names=c("C","S","W","Sa","Wt"))
```

Now I get the number of observations using `aggregate`. The new data frame called `plot.table.4.3` contains `Sa` (number of satellites), `W` (width), and the number of observations at that Sa, W combination.

```
plot.table.4.3<-aggregate(rep(1,nrow(table.4.3)), list(Sa=table.4.3$Sa,
   W=table.4.3$W), sum)
```

As `aggregate` will change `Sa` and `W` to factors, we must convert them back to numeric.

```
plot.table.4.3 <- convert.col.type(target = plot.table.4.3, column.spec = list("Sa"),
   column.type = "double")
#faster: plot.table.4.3$Sa<-as.numeric(levels(plot.table.4.3$Sa))[plot.table.4.3$Sa]
# R: plot.table.4.3$Sa<-as.numeric(as.vector(plot.table.4.3$Sa))
plot.table.4.3 <- convert.col.type(target = plot.table.4.3, column.spec = list("W"),
   column.type = "double")
# R: plot.table.4.3$W<-as.numeric(as.vector(plot.table.4.3$W))
```

Now, I plot figure 4.3 (plot not shown)

```
plot(y=plot.table.4.3$Sa,x=plot.table.4.3$W,xlab="Width (cm)",
ylab="Number of Satellites", bty="L", axes=F, type="n")
axis(2, at=1:15)
axis(1, at=seq(20,34,2))
text(y=plot.table.4.3$Sa,x=plot.table.4.3$W,labels=plot.table.4.3$x)
```

It is probably possible to do the above using `table` instead of `aggregate`.

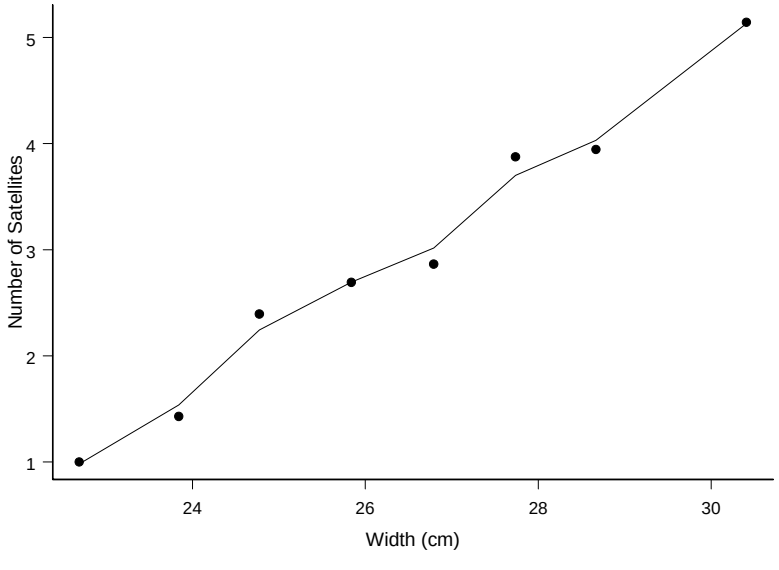
Figure 4.4 plots the mean number of satellites for the mean width of eight width categories created from the `W` variable. In addition, a fitted smooth from the result of a generalized additive model (GAM) fit to these data is superimposed. Generalized additive models (as extensions to GLIMs) are discussed briefly in Section 4.8 in Agresti, and in the corresponding section of this manual.

In the following code, first I cut the `W` variable into categories, then use `aggregate` again to get means of `Sa` and of `W` by the categories. Then, the points are plotted.

```
table.4.3$W.fac<-cut(table.4.3$W, breaks=c(0,seq(23.25, 29.25),Inf))
plot.y<-aggregate(table.4.3$Sa, by=list(W=table.4.3$W.fac), mean)$x
plot.x<-aggregate(table.4.3$W, by=list(W=table.4.3$W.fac), mean)$x
plot(x=plot.x, y=plot.y, ylab="Number of Satellites", xlab="Width (cm)",bty="L",
axes=F, type="p", pch=16)
axis(2, at=0:5)
axis(1, at=seq(20,34,2))
```

Next, I use `gam` in S-PLUS (and `gam` from package `mgcv` in R) to fit the GAM using a Poisson distribution with log link. The smooth that is used for the width means allows for five degrees of freedom (default was three). This smooth is distinctly different from the smooth in Figure 4.4 in Agresti, as it falls between the points and not directly below it.

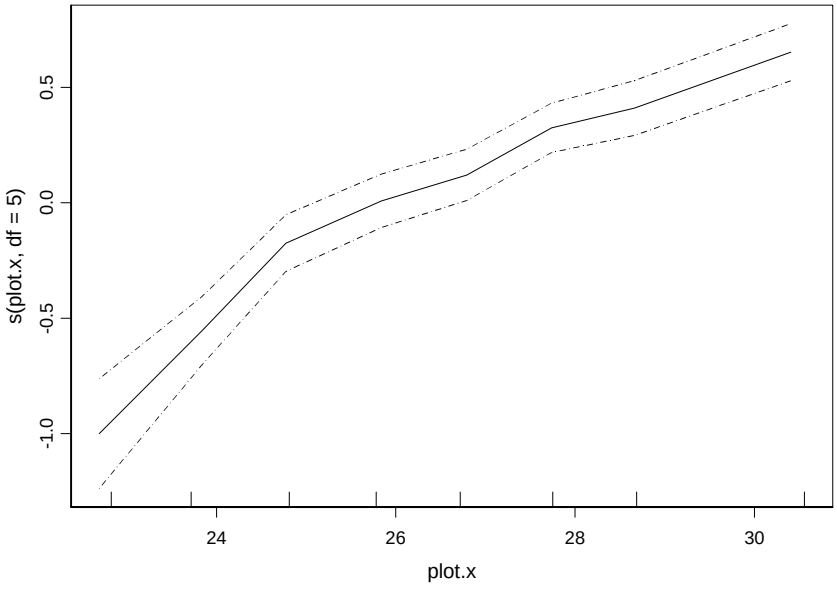
```
res<-gam(plot.y~s(plot.x, df=5), family=poisson(link=log))
# R: library(mgcv)
# res<- gam(plot.y~s(plot.x, k=4, fx=TRUE), family=poisson(link=log))
lines(x=plot.x,y=res$fitted.values)
```



If we set `df=1` or `1.5` in `s`, we get a smooth that curves slightly upward at both ends.

To see exactly what the smoothed term in width in the `gam` call represents, we can plot (mean) width by the smoothed term. The `se=T` argument adds plus and minus two pointwise standard deviations as dashed lines to the plot.

```
plot.gam(res, se=T)
```



The smoothed term does not deviate too much from linearity, and Figure 4.4 shows a linear trend relating the mean number of satellites to width. Agresti fits Poisson GLIMs with both log and identity links to the data.

We will use IRLS and `glm` to fit the Poisson models. For non-canonical links (e.g., identity), the estimates may differ slightly from Agresti's. A Poisson loglinear model is fit using

```
log.fit<-glm(Sa~W, family=poisson(link=log),data=table.4.3)
summary(log.fit)

Call: glm(formula = Sa ~ W, family = poisson(link = log), data = table.4.3)
Deviance Residuals:
       Min         1Q     Median         3Q        Max
 -2.852632 -1.988425 -0.4933188 1.09697 4.922148

Coefficients:
                 Value Std. Error   t value
(Intercept) -3.3047572 0.54222774 -6.094777
          W  0.1640451 0.01996492  8.216665

(Dispersion Parameter for Poisson family taken to be 1 )

    Null Deviance: 632.7917 on 172 degrees of freedom

Residual Deviance: 567.8786 on 171 degrees of freedom

Number of Fisher Scoring Iterations: 5

Correlation of Coefficients:
  (Intercept)
W -0.996627
```

I've given the output from a call to `summary` (in S-PLUS), which summarizes model fitting by giving coefficient estimates, their approximate SEs, residual deviance, and a summary of deviance residuals. The dispersion parameter by default is set to 1 (we will change that later). The correlation between the coefficient estimates is quite high (almost perfectly negative). This can be reduced by using the mean deviation in width instead of width in the model. The null deviance is the deviance value (p. 119, Agresti) for a model with only an intercept. The residual deviance is the deviance value for a model with both an intercept and width. The reduction in deviance is a test of the width coefficient. That is

```
log.fit$null.deviance-log.fit$deviance
[1] 64.91309
```

is the LR statistic for testing the significance of the Width variable in the model. Compared to a chi-squared distribution with 1 degree of freedom, the p-value of this test is quite low, rejecting the null hypothesis of a zero-valued coefficient on width. We can get similar information from the Wald test given by the t-value next to the coefficient estimate (z-value in R version). However, the LRT is usually considered more reliable (see Agresti, and also Lloyd 1999).

The `summary` result for any `glm.object` in S-PLUS has the following attributes that can be extracted:

```
attributes(summary(log.fit)) # S-PLUS
$names:
 [1] "call"           "terms"          "coefficients"   "dispersion"     "df"    "deviance.resid"
 [7] "cov.unscaled"   "correlation"    "deviance"       "null.deviance"  "iter"           "nas"
[13] "na.action"

$class:
[1] "summary.glm"
```

The same summary call in R has a few additional components, notably AIC.

```
attributes(summary(log.fit)) # R

$names
 [1] "call"           "terms"          "family"         "deviance"
 [5] "aic"            "contrasts"      "df.residual"    "null.deviance"
 [9] "df.null"        "iter"           "deviance.resid" "aic"
[13] "coefficients"   "dispersion"     "df"             "cov.unscaled"
[17] "cov.scaled"
```

As an example, to extract the estimated coefficients, along with their standard errors, type:

```
summary(log.fit)$coefficients

                Value Std. Error   t value
(Intercept) -3.3047572 0.54222774 -6.094777
          W  0.1640451 0.01996492  8.216665
```

Thus, the fitted model is

$$\log \hat{\mu} = -3.305 + 0.164\,x \qquad (4.2)$$

The `glm.object` itself has the following components. It "inherits" all the attributes of `lm.objects`

```
attributes(log.fit) # S-PLUS (R has quite a few more)
$names:
 [1] "coefficients"     "residuals"         "fitted.values"     "effects"         "R"
 [6] "rank"             "assign"            "df.residual"       "weights"    "family"
[11] "linear.predictors" "deviance"          "null.deviance"     "call"        "iter"
[16] "y"                "contrasts"         "terms"             "formula"   "control"

$class:
[1] "glm" "lm"
```

For example, the fitted response values (expected values) at each of the width values in the model can be obtained by extracting the `fitted.values`.

```
log.fit$fitted.values
```

The same answer can be obtained by the call **`fitted(log.fit)`**. The functions, `fitted`, `resid`, `coef` are shortened versions to extract fitted values, residuals, and coefficients, respectively, from `glm` objects.

Using the `predict` method for `glm`, we can get the fitted response value for *any* width value we input. For example, the expected number of satellites at a width of 26.3 is

```
predict.glm(log.fit, type="response", newdata=data.frame(W=26.3))

[1] 2.744581
```

Agresti also fits a Poisson model with identity link to the Horseshoe Crab data. This is fit in S-PLUS using

```
id.fit<-glm(Sa~W, family=poisson(link=identity),data=table.4.3, start=predict(log.fit,
   type="link")) # S-PLUS

summary(id.fit)$coefficients

Coefficients:
                  Value Std. Error   t value
(Intercept) -11.4051613 0.99511013 -11.46121
          W   0.5446717 0.04056604  13.42679
```

Because of convergence problems with the identity link, we give the initial estimate of the log of the mean vector μ as the estimated linear predictor vector η from the log link fit, instead of using the (log of the) data values, which is the default (see bottom of p.147 of Agresti). In R, use the estimated coefficients themselves. Thus,

```
# R
id.fit<-glm(Sa~W, family=poisson(link=identity),data=table.4.3, start=coef(log.fit))
```

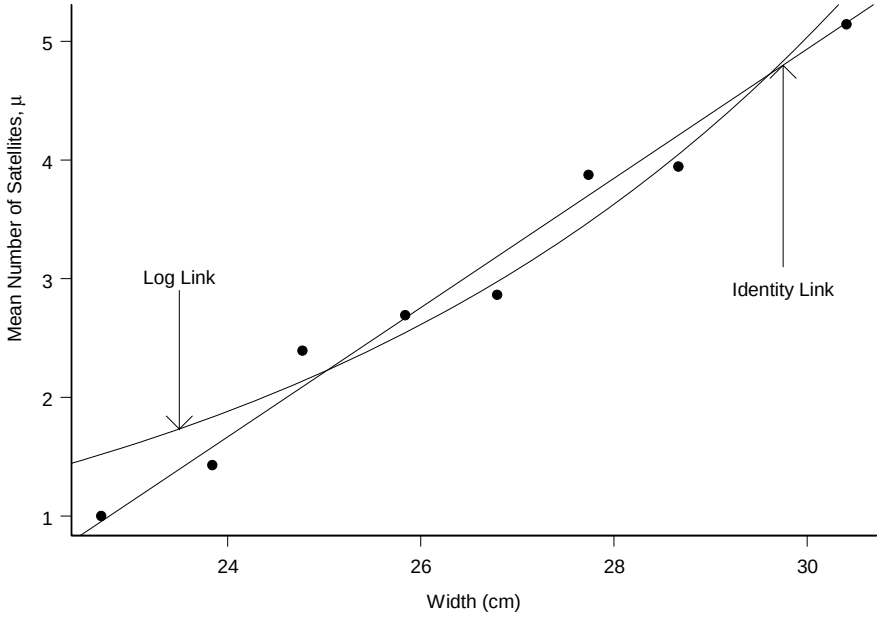
The fitted model is then

$$\hat{\mu} = -11.41 + 0.55x \tag{4.3}$$

Thus, carapace width has an additive impact on mean number of satellites instead of multiplicative, as with the log link. The additive effect is 0.55 (about half a satellite) per 1 cm increase in width. A comparison of the two models' predictions is in Figure 4.5 in Agresti, which is reproduced below. I use the `guiCreate` function (in S-PLUS only) to make the Greek letter mu. (In R, there is a flexible package called `plotmath` (in the "base" environment) that can be used to create mathematical notation. See below and `help(plotmath)`). I also use the `arrows` command to make arrows. This function is somewhat different in S-PLUS and R (most notably the coordinate arguments!).

```
plot(x=plot.x, y=plot.y, ylab="", xlab="Width (cm)",bty="L",axes=F, type="p", pch=16)  #
# R: plot(x=plot.x, y=plot.y, ylab=expression(paste("Mean number of satellites,",
   {mu})), xlab="Width (cm)",bty="L",axes=F, type="p", pch=16)
axis(2, at=0:5)
axis(1, at=seq(20,34,2))

# make y-axis title (Only needed in S-PLUS.  See plot call for R above.)
guiCreate( "CommentDate", Name = "GSD2$1",
   Title = "Mean Number of Satellites, \\\"Symbol\"m",
   FillColor = "Transparent", FontSize="16")
```

```
guiModify( "CommentDate", Name = "GSD2$1",
    xPosition = "0.542857",
    yPosition = "3.52967", Angle = "90")

# make arrows and text
ind<-order(table.4.3$W)
lines(x=table.4.3$W[ind],y=log.fit$fitted.values[ind])
lines(x=table.4.3$W[ind],y=id.fit$fitted.values[ind])
arrows(x1=23.5,y1=2.9,x2=23.5,y2=predict(log.fit,newdata=data.frame(W=23.5),
    type="response"), open=T, size=.3)
# R: arrows(x0=23.5,y0=2.9,x1=23.5,y1=predict(log.fit,newdata=data.frame(W=23.5),
    type="response"), length=.2)
text(x=23.5,y=3,"Log Link")
arrows(x1=29.75,y1=3.1,x2=29.75,y2=predict(id.fit,newdata=data.frame(W=29.75),
    type="response"), open=T, size=.3)
# R: arrows(x0=29.75,y0=3.1,x1=29.75,y1=predict(id.fit,newdata=data.frame(W=29.75),
    type="response"), length=.2)
text(x=29.75,y=2.9,"Identity Link")
```



Using the R version of `summary.glm` (or the `extractAIC` function in `MASS` library in S-PLUS), we can compare the AICs of these two link functions. The returned AIC assumes that the dispersion parameter is known; so there is a caveat in using it when the dispersion parameter is actually estimated (see next section).

```
summary.glm(log.fit)$aic   # R
[1] 927.1762

summary.glm(id.fit)$aic # R
[1] 917.014
```

A lower AIC implies a better model fit. A comparison of residuals from each of the models may also be helpful.

D. Overdispersion in Poisson Generalized Linear Models

If we examine Table 4.4 (p. 129 in Agresti) of the sample mean and variance of the number of satellites per Width category, we see that the variance exceeds the mean in all cases. A Poisson model postulates that for each Width, the variance and mean of the number of satellites is the same. Thus, we have evidence of overdispersion, where the "ordinary" Poisson model is not correct. Incidentally, the function `tapply` (or `by`) can be used to get the sample means and variances:

```
tapply(table.4.3$Sa, table.4.3$W.fac, function(x) c(mean=mean(x), variance=var(x)))
# by(table.4.3$Sa, table.4.3$W.fac, function(x) c(mean=mean(x), variance=var(x)))
```

The general density for the random component of a GLIM has an unspecified dispersion parameter that is related to the scale of the density. This density is given in equation (4.14) in Agresti (2002). For example, with a Gaussian random component, the dispersion parameter is the variance. For a binomial or Poisson random component, the dispersion is fixed at 1, so we have not needed to deal with it yet. However, if we believe that the mean response is like a Poisson mean, but the variance is proportional to the Poisson mean (with proportionality constant the dispersion parameter), then we can leave the dispersion parameter unspecified and estimate it along with the mean parameter. In this case, we no longer have a Poisson distribution as our random component. Plus, the estimate of the dispersion parameter must be used in standard error and deviance calculations. The general form of the scaled deviance is given in equation (4.30) in Agresti.

If the actual Poisson means (i.e., per Width value) are large enough (or, in the case of binomial data, the sample sizes at each covariate condition are large enough), then a simple way to detect overdispersion is to compare the residual deviance with the residual df, which will be equal in the asymptotic sense just mentioned. These two quantities are quite disparate for both link models. So, we decide to estimate the dispersion parameter instead of leaving it fixed at 1. We can do this in S-PLUS using a `dispersion=0` argument to `summary.glm`. In R, we must use the `quasipoisson` family first. (This makes sense after the discussion of quasi-likelihood).

```
summary.glm(log.fit, dispersion=0)$dispersion  # S-PLUS

  Poisson
 3.181952

# R
log.fit.over<-glm(Sa~W, family=quasipoisson(link=log),data=table.4.3)
summary.glm(log.fit.over)$dispersion

[1] 3.181786
```

The estimate of the dispersion parameter is the sum of the squared Pearson residuals divided by the residual degrees of freedom. The rationale for this is given on p. 150 in Agresti, and is based on moment estimation. From the estimate given, the variance of our random component (the number of satellites for each Width) is roughly three times the size of the mean.

Other estimates returned by `summary.glm` are adjusted using the dispersion estimate. For example, the standard errors are multiplied by it. Compare the following in S-PLUS:

```
res<-summary.glm(log.fit)
sqrt(diag(res$cov.unscaled) * summary.glm(log.fit, dispersion = 0)$dispersion) # see
   also vcov.glm from MASS

[1] 0.96722735 0.03561348

summary.glm(log.fit, dispersion = 0)$coefficients

                Value Std. Error   t value
(Intercept) -3.3047572 0.96722735 -3.416733
          W  0.1640451 0.03561348  4.606263
```

The usual deviance must be divided by the dispersion estimate to get the scaled deviance used in F-tests for model comparison (see Venables and Ripley, 2002). An `anova` method is available for `glm` objects that does these tests provided argument `test="F"` is supplied. `anova.glm` will print the F-test that uses the dispersion estimate from the larger model. Thus, if we wanted to compare the null model (with $q = 1$ parameters) to `log.fit` (with $p = 2$) using the F-test (in Agresti's notation)

$$F = \frac{(D(\mathbf{y};\boldsymbol{\mu}_0) - D(\mathbf{y};\boldsymbol{\mu}_1))}{(p-q)\hat{\phi}} \tag{4.4}$$

we would do

```
null.fit<-glm(Sa~1, family=poisson(link=log),data=table.4.3)
anova.glm(null.fit, log.fit,test="F")   # S-PLUS
# R: anova.glm(null.fit, log.fit.over, test="F")

Analysis of Deviance Table

Response: Sa

  Terms Resid. Df Resid. Dev Test Df Deviance F Value         Pr(F)
1     1       172   632.7917
2     W       171   567.8786       1 64.91309 20.4004 0.00001166917
```

Note that this is the correct F value because

```
(deviance(null.fit)-deviance(log.fit))/summary.glm(log.fit, dispersion=0)$dispersion
# R (deviance(null.fit)-deviance(log.fit))/summary.glm(log.fit.over)$dispersion
[1] 20.4004
```

Because the estimate of dispersion is based on a chi-squared statistic that has an approximate chi-squared distribution (see p. 150 in Agresti), it is a good idea to meet the assumptions for this approximation in terms of number of cases per unique covariate combination. Thus, Agresti uses the satellite totals and fit for all female crabs at a given width to increase the counts and fitted values.

In S-PLUS or R, the chi-squared statistic is

```
Sa<-tapply(table.4.3$Sa, table.4.3$W, sum)
mu<-tapply(predict(log.fit, type="response"), table.4.3$W, sum)
(chi.squared<-sum(((Sa-mu)^2)/mu))
[1] 174.2737
```

with estimated dispersion parameter

```
chi.squared/64
[1] 2.723027
```

From the revised estimate, the variance of our random component (the number of satellites for each Width) is a little less than three times the size of the mean. As a Poisson random component assumes that the variance and mean are equal, an ordinary Poisson random component is probably not a good model choice. In Chapter 12, Agresti discusses adding random effects to GLIMs, which can help account for overdispersion. Another possibility is to fit a GLIM with negative binomial random component, as described below. The negative binomial does not assume the variance and mean are equal.

E. Negative Binomial GLIMs

For count data, a negative binomial random component has a second parameter in additional to the mean, called the dispersion parameter. The variance of the random component is a function of both the

mean and dispersion parameter. The pdf of the negative binomial is given in equation (4.12) of Agresti, with

$$E(Y) = \mu \text{ and } \text{var}(Y) = \mu + \mu^2 / k$$

where k is the (positive) dispersion parameter. The smaller the dispersion parameter, the larger the variance as compared to the mean. But, with growing dispersion parameter, the variance converges to the mean, and the random quantity converges to the Poisson distribution. Actually, negative binomial is the distribution of a Poisson count with gamma-distributed rate parameter (see Section 13.4 in Agresti).

There are several methods for going about fitting a negative binomial GLIM in S-PLUS and R. If the dispersion parameter is known, then the MASS library has a `negative.binomial` family function to use with `glm`. For example, to fit a negative binomial GLIM to the Horseshoe Crab data, where the dispersion parameter (called `theta` in the function) is *fixed* to be 1.0, we can use

```
library(MASS)
glm(Sa~W, family=negative.binomial(theta=1.0,link="identity"),data=table.4.3,
    start=predict(log.fit, type="link")) # for R use start=coef(log.fit)

Call:
glm(formula = Sa ~ W, family = negative.binomial(theta = 1, link = "identity"), data =
    table.4.3, start = predict(log.fit, type = "link"))

Coefficients:
 (Intercept)         W
   -11.62843 0.5537723

Degrees of Freedom: 173 Total; 171 Residual
Residual Deviance: 202.8936
```

A simpler version of `negative.binomial` (called `neg.bin`, with only the log link) is also available from MASS, as well as a function called `glm.nb`. The function `glm.nb` allows one to estimate `theta` using maximum likelihood estimation. The output is similar to that of `glm` and has a `summary` method.

```
library(MASS)
nb.fit<-glm.nb(Sa ~ W, data = table.4.3, init.theta=1.0, link=identity,
    start=predict(id.fit, type="link"))  # for R use start=coef(id.fit)
summary(nb.fit)

Call: glm.nb(formula = Sa ~ W, data = table.4.3, start = predict(id.fit, type =
    "link"), init.theta =  0.931699963253856, link = identity)

Deviance Residuals:
      Min        1Q     Median        3Q      Max
 -1.78968 -1.409158 -0.2558914 0.4522508 2.106918

Coefficients:
                  Value Std. Error   t value
(Intercept) -11.6329804 1.08204466 -10.75092
          W   0.5539562 0.05135274  10.78728

(Dispersion Parameter for Negative Binomial family taken to be 1 )

    Null Deviance: 216.5127 on 172 degrees of freedom

Residual Deviance: 195.5161 on 171 degrees of freedom

Number of Fisher Scoring Iterations: 1

Correlation of Coefficients:
  (Intercept)
W -0.9996673
```

```
              Theta:  0.932
          Std. Err.:  0.168

 2 x log-likelihood:  -747.929
```

The coefficient estimates are slightly different that those obtained by Agresti (p. 131).

LRTs are also available for comparing nested negative binomial GLIMs (via the `anova` method). However, as theta must be held constant between the two models being compared, one must first convert the `glm.nb` object to a `glm` object, which will fix theta if the model is then `update`'d. The conversion is done using the `glm.convert` function.

To use custom GLIMs, one must create a family. The function `make.family` can be used to make a new `glm` family. You must specify the name of the family, the link functions allowed for it (e.g., `logit`, `log`, `cloglog`) and their derivatives, inverses, and initialization expressions (used to initialize the linear predictor for Fisher scoring), and the variance and deviance functions.

For example, in the function listing for `negative.binomial`, you can see where the links and variances lists are defined and where the function `make.family` is used. The reason why the link functions are not actually typed out in function form is because all the links already appear in other `glm` families in S-PLUS. Their information is stored in the matrix `glm.links`. `negative.binomial` accesses the appropriate link function using `glm.links[,link]`.

Beta-binomial regression models can be fit using `gnlr` in library `gnlm` for R.

F. Residuals for GLIMs

To demonstrate how to obtain the residuals in Agresti's Section 4.5.5, I use the Horseshoe Crab data fit. For example, the Pearson and deviance residuals are obtained from

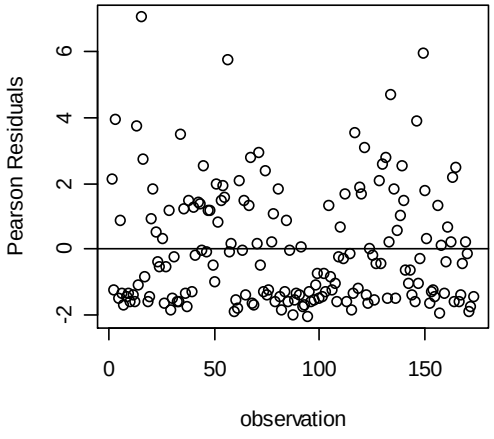
```
resid(log.fit, type="deviance")
pear.res<-resid(log.fit, type="pearson")
```

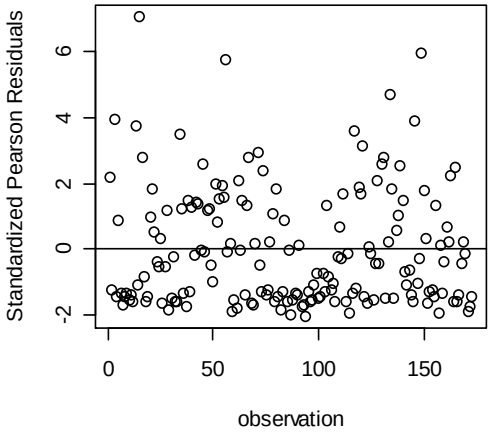
The standardized Pearson residuals are obtained by dividing the output from `resid` by a function of the hat diagonal values, obtained from the `lm.influence` function.

```
pear.std<-resid(log.fit, type="pearson")/sqrt(1-lm.influence(log.fit)$hat) # both S-
    PLUS and R
```

However, as most of the hat diagonals are very small, there is really no difference between the two sets of residuals, as the following graph shows.

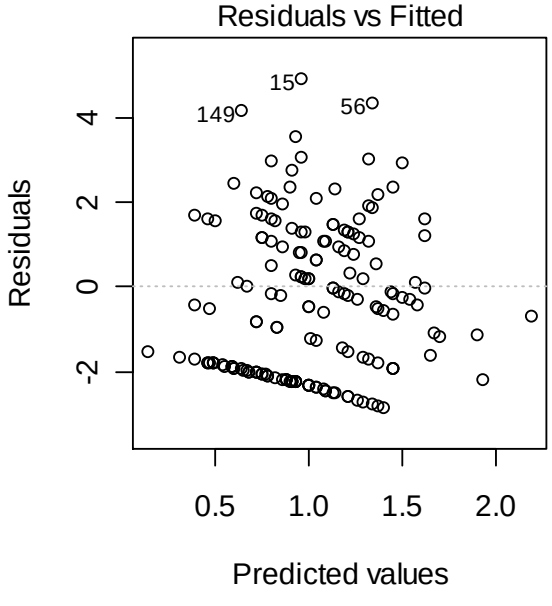
```
par(mfrow=c(2,2))
plot(pear.res, xlab="observation",ylab="Pearson Residuals")
abline(h=0)
plot(pear.std, xlab="observation",ylab="Standardized Pearson Residuals")
abline(h=0)
```

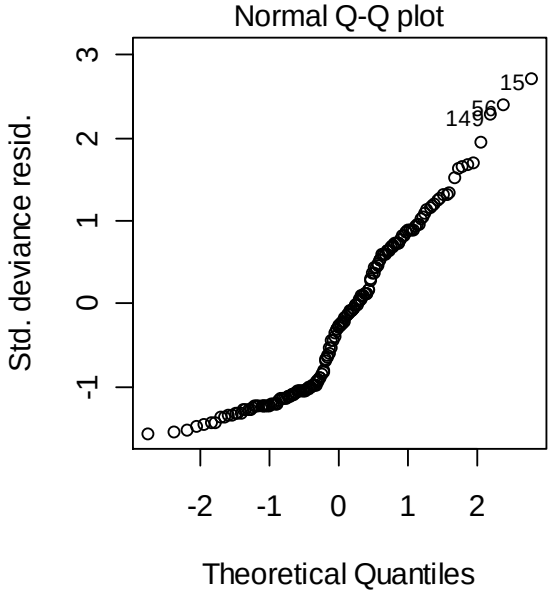


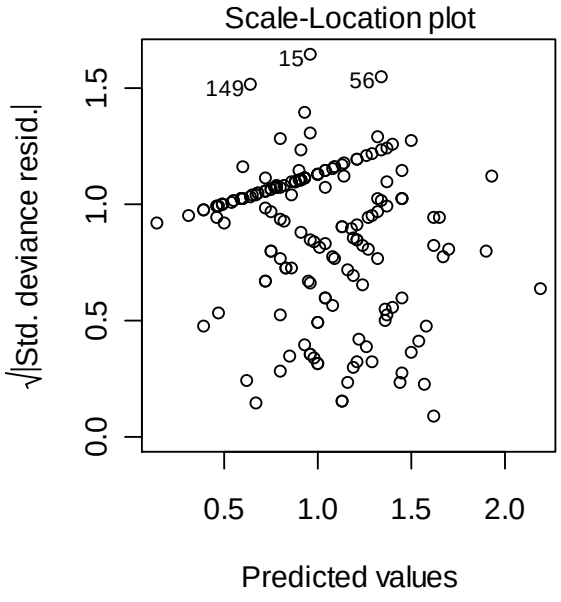


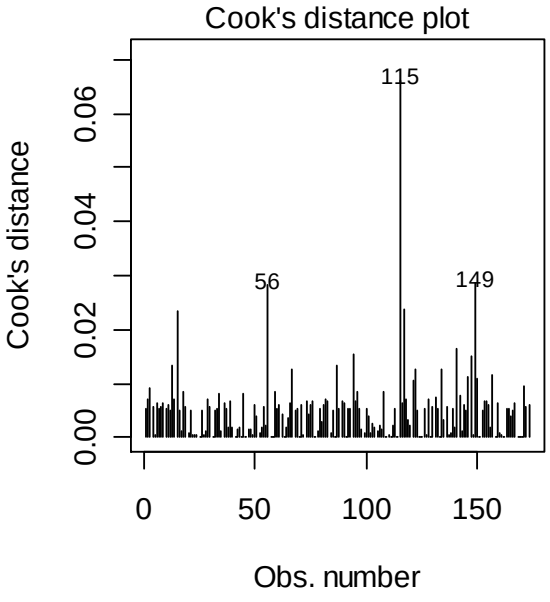
Issuing the command `plot(log.fit)` may also be useful:

```
par(mfrow=c(1,2))
old.par<-par(pty="s")   # save old pty settings
par(pty="s")            # change pty par setting to "s"
plot(log.fit)
par(old.par)            # change back to old pty settings
```









G. Quasi-Likelihood and GLIMs

Quasi-likelihood estimation assumes only a mean-variance relationship rather than a complete distribution for the response variable. However, many times the relationship specified determines a particular distribution. This is because distributions in the natural exponential family are characterized by the relationship between the mean and the variance function. Thus, using the R function `quasipoisson` with `log` link (and `constant` variance) gives the same estimates as using the Poisson family with log link because the Poisson is a distribution in the natural exponential family. The reason the estimates are the same is because the estimating equations used for quasi-likelihood estimation are the same as the likelihood equations in the case of a specified distribution with that mean and variance relationship.

In general then, quasi-likelihood estimates (QLEs) are not MLEs, but in some cases where we are assuming a particular distribution, but with a magnified variance (e.g., when the dispersion parameter multiplies the variance), the point estimates will be identical because the dispersion parameter drops out of the estimating equations.

Agresti uses data from a teratology experiment to illustrate overdispersion and the use of quasi-likelihood estimation. Rats in 58 litters were on iron-deficient diets and given one of four treatments (groups 1-4). The rats were made pregnant and killed after three weeks. The number of dead fetuses out of the total litter size is the response variable. (The data set is available on Agresti's CDA website). I copied the data into a text file called `teratology.ssc`. It is then read into S-PLUS/R by

```
table.4.5<-read.table("teratology.ssc",
   col.names=c("","group","litter.size","num.dead"))[,-1]
table.4.5$group<-as.factor(table.4.5$group)
```

I changed the group column to be treated as a factor instead of numeric.

The response is assumed binomially distributed. Initially, the probability of death is assumed to differ only across treatment groups, but is identical for all litters within a treatment group. This model can be fit via maximum likelihood in S-PLUS/R using `glm` by removing the intercept term in the model formula. In this way, we get an identity link

```
fit1<-glm(num.dead/litter.size~group-1, weights=litter.size, data=table.4.5,
   family=binomial)
```

The MLEs of the probabilities are then

```
(pred<-unique(round(predict(fit1, type="response"),3)))
[1] 0.758 0.102 0.034 0.048
```

with SEs

```
(SE<-sqrt(pred*(1-pred)/tapply(table.4.5$litter.size,table.4.5$group,sum)))
          1           2           3           4
 0.02368473 0.02786104 0.02379655 0.0209615
```

Pearson's chi-squared statistic gives

```
(chi.squared<-sum(resid(fit1, type = "pearson")^2))
[1] 154.7069
```

which compared to the residual degrees of freedom (58 – 4) = 54 is quite large. This may indicate overdispersion, although the fitted counts ($n_{i(g)}\hat{\pi}_g$'s) are not all that large. To adjust for possible overdispersion, Agresti uses the square root of the estimate of the dispersion parameter to multiply the SEs.

```
SE*sqrt(chi.squared/54)
          1           2           3           4
 0.04008367 0.04715159 0.04027292 0.03547493
```

We could have found the same estimates using quasi-likelihood estimation. In particular, the following gives the probability estimates as the coefficient estimates. I used a `quasi` family with `identity` link and with `variance` related to the mean, `mu`, as `mu(1-mu)`. Because I used an identity link, I first get starting values from a previous fit (`fit1`).

```
glm(num.dead/litter.size~group-1, weights=litter.size, data=table.4.5,
    family=quasi(link=identity, variance="mu(1-mu)"),
    start=predict(fit1, type="response"))
# R: glm(num.dead/litter.size~group-1, weights=litter.size, data=table.4.5,
family=quasi(link=identity, variance="mu(1-mu)"), start=unique(predict(fit1,
type="response")))


Coefficients:
    group1    group2     group3     group4
 0.7584098 0.1016949 0.03448276 0.04807692

Degrees of Freedom: 58 Total; 54 Residual
Residual Deviance: 173.4532
```

For R, we need to specify starting values in the range from 0 to 1, as the coefficients will be probabilities. Thus, I take the (unique set of) predicted probabilities from `fit1` to use as starting values.

H. Generalized Additive Models (GAMs)

As the name implies, GAMs generalize additive models in the same way that GLIMs generalize linear models. GAMs extend GLIMs by allowing the predictors to enter the model in flexible ways, via a smooth function that is not necessarily linear or some other simple transformation like logarithmic. Thus, the link function of the mean response is

$$g(\mu_i) = \sum_j s_j(x_{ij}) \tag{4.5}$$

where the s_j are smooth functions of the predictors. When the s_j are linear, the GAM becomes a GLIM.

The functions s_j are fit using scatterplot smoothers. One example of a scatterplot smoother is locally weighted least squares regression (*lowess*). Another example is a smoothing spline. A common function for the s_j is a restricted cubic spline. A cubic spline in a predictor variable is a piecewise cubic polynomial representation of the predictor, where the pieces are determined by "knot" locations placed over the range of the predictor variable. The restriction occurs by forcing the resulting piecewise curve to have vanishing second derivatives at the boundaries. Then the number of degrees of freedom for each original predictor variable is equal to the number of knots minus 1. The more degrees of freedom allotted to a predictor variable, the more complex the relationship between the predictor and the response.

A Fisher scoring-type outer loop (to update the working responses) with a backfitting inner loop (to update the smooth functions) can be used to find the estimates of the s_j (see Chambers and Hastie, 1992, p. 300ff). When we assume that the s_j in a GAM are polynomial smoothing splines (like the cubic splines above), then the algorithm corresponds to penalized maximum likelihood estimation, where the penalty is for roughness of the s_j (Green and Silverman, 1994).

The function `gam` in S-PLUS can be used to fit GAMs. If the functions are restricted to be smoothing splines (denoted by `s` in the `gam` formula) then `gam` uses a basis function algorithm, with a basis of cubic B-splines, to find the smooth functions (see Green and Silverman, 1994, p. 44ff). The number of knots covering the range of the predictor values is chosen by default to be the number of unique data points (if these are less than 50), and otherwise "a suitable fine grid of knots is chosen". The smoothing parameter used in the penalized least squares estimation to find the basis function representation of the smooth functions can be specified by the user or determined via generalized cross-validation (see Green and Silverman, 1994, p. 35). Equivalently, the number of degrees of freedom for a smooth function can be specified.

The function `gam` in S-PLUS "inherits" a lot of functionality from `glm` and `lm`. But, the output is more graphical because the fitted smooth functions usually must be graphed to interpret the model (Chambers and Hastie, 1992).

Horseshoe Crab Data – Bernoulli response

Agresti fits a GAM to the binary response of whether a female crab has at least one satellite, using a logit link and the width of carapace as a predictor. We fit a GAM using `gam` in S-PLUS and R, with 3 degrees of freedom for width. Then, we plot the fitted smooth function.

First, I get the binary response and the number of observations at each data point.

```
table.4.3$Sa.bin<-ifelse(table.4.3$Sa>0,1,0)
plot.table.4.3<-aggregate(table.4.3$Sa,
   by=list(Sa.bin=table.4.3$Sa.bin,W=table.4.3$W), length)
plot.table.4.3 <- convert.col.type(target = plot.table.4.3, column.spec =
   list("Sa.bin"), column.type = "double")
# R: plot.table.4.3$Sa.bin<-as.numeric(as.vector(plot.table.4.3$Sa.bin))
plot.table.4.3 <- convert.col.type(target = plot.table.4.3, column.spec = list("W"),
   column.type = "double")
# R: plot.table.4.3$W<-as.numeric(as.vector(plot.table.4.3$W))
```

Plot the number of observations.

```
old.par<-par(pty="s")  # save previous pty setting (use par(old.par) later to reset
   previous setting)
par(pty="s")           # change pty par setting to "s"
plot(y=table.4.3$Sa.bin,x=table.4.3$W,xlab="Width, x (cm)",
ylab="Probability of presence of satellites", axes=F, type="n")
# R: plot(y=table.4.3$Sa.bin,x=table.4.3$W,xlab=expression(paste("Width, ", italic(x),
   "(cm)")), ylab="Probability of presence of satellites", axes=F, type="n")
axis(2, at=c(0,1))
axis(1, at=seq(20,34,2))
text(y=plot.table.4.3$Sa.bin,x=plot.table.4.3$W,labels=plot.table.4.3$x, cex=.5)
guiModify( "XAxisTitle", Name = "GSD2$1$Axis2dX1$XAxisTitle",
```

```
    Title = "Width, `x` (cm)") # S-PLUS only (GSD2 is the name of the graphsheet.
    Substitute the name of your graphsheet in place of it, if GSD2 is not it)
```

Fit the GAM and plot the fit.

```
res<-gam(Sa.bin~s(W, df=3), family=binomial(link=logit), x=T, data=plot.table.4.3)
# R: library(mgcv)
# R: res<-gam(Sa.bin~s(W, k=3, fx=TRUE, bs="cr"), family=binomial(link=logit),
    data=plot.table.4.3)
lines(x=plot.table.4.3$W,y=res$fitted.values)
```

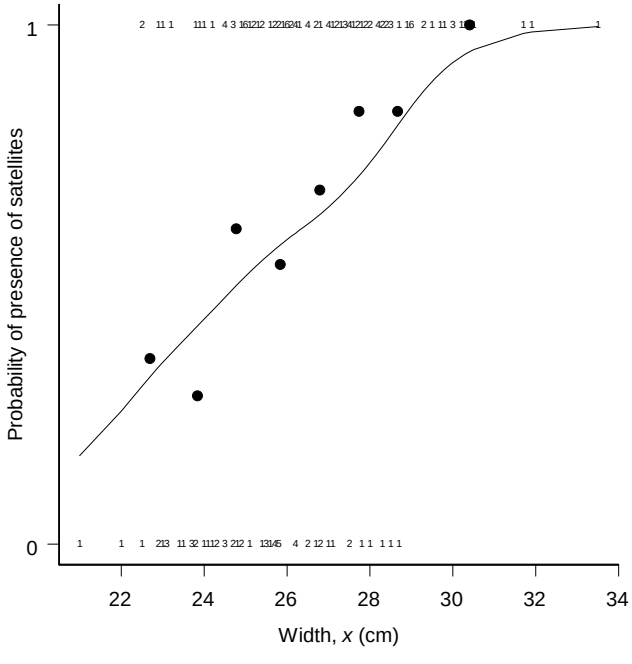
Get the proportions within each width category. Width categories were defined earlier in Section C of this manual.

```
prop<-aggregate(table.4.3$Sa.bin, by=table.4.3$W.fac, mean)$x
# R, must be: prop<-aggregate(table.4.3$Sa.bin, by=list(W=table.4.3$W.fac), mean)$x
```

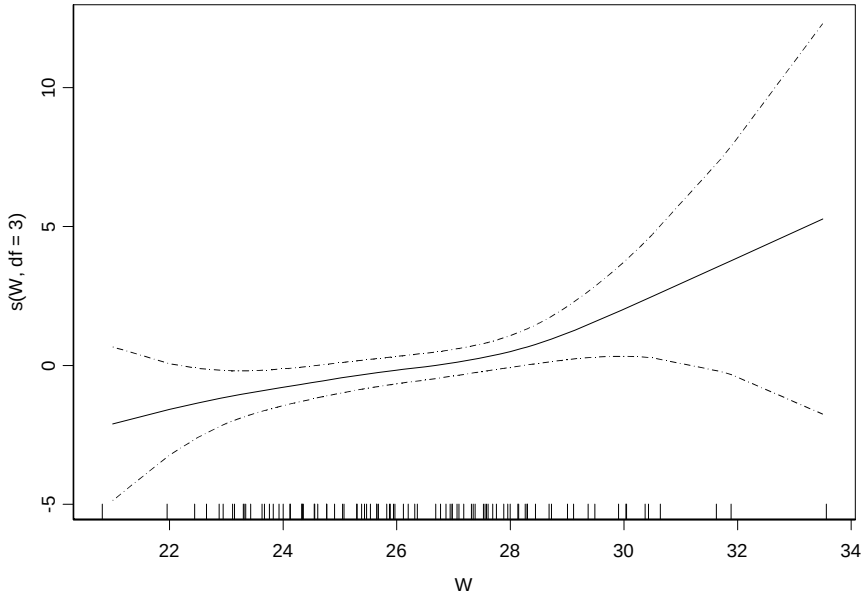
Now, put the proportions on the plot.

```
lines(plot.x, prop, type="p",pch=16) # see above for defn of plot.x
```

The figure shows that an S-shaped function may describe the data well. This signifies that a logistic regression model may be appropriate. Indeed, the `plot.gam` function shows that the smoothing spline in width is close to linear for at least the middle portion of width values.



```
plot.gam(res, se=T)
```




```
par(old.par)              # change back to old pty setting
```

Chapter 5: Logistic Regression

A. Summary of Chapter 5, Agresti

This chapter treats logistic regression in more detail than did Chapter 4. It begins with univariate logistic regression. The interpretation of the coefficient, β, in the univariate logistic regression (equation 5.1 in Agresti) is discussed initially. The direction of this coefficient indicates the direction of the effect of the variable x on the probability of a positive response. The magnitude of β is usually interpreted in terms of the odds of a positive response. The odds change multiplicatively by $\exp(\beta)$ for each one-unit increase in x. The expression $\exp(\beta)$ is actually an odds ratio: the odds (of positive response) at $X = x+1$ to the odds at $X = x$. Prior to fitting a logistic regression model to data, one should check the assumption of a logistic relationship between the response and explanatory variables. A simple way to do this is to use the linear relationship between the logit and the explanatory variable. The values of the explanatory variable can be plotted against the sample logits (p. 168, Agresti) at those values. The plot should look roughly linear for a logistic model to be appropriate. If there are not enough response data at each unique x value (and categorizing x values is undesirable), then the technique of the last section in Chapter 4 can be used (i.e., GAM). There, we saw that a sigmoidal (or S-shaped) trend appeared in the plot of the response by predictor (Figure 4.7, Agresti).

Logistic regression can be used with retrospective studies to estimate odds ratios. The fit of a logistic regression model to retrospective response data, given an explanatory variable whose values are not known in advance, yields a coefficient estimate whose exponent is the same estimated odds ratio as if the response variable had been prospective.

A logistic regression model is fit via maximum likelihood estimation. In practice, this can be achieved via IRLS, as mentioned in the previous chapter. That is, the MLE is the limit of a sequence of weighted least squares estimates, where the weight matrix changes at each iteration (see Section 5.5 in Agresti).

Inference for the maximum likelihood estimators is asymptotic. Confidence intervals can be Wald confidence intervals, LR confidence intervals or score confidence intervals. The Wald, LR, and score tests can be used to test hypotheses. The LRT is preferred, as it uses both the null maximized likelihood value as well as the alternative maximized likelihood value (providing more information than the other tests), instead of just one of these values. When comparing two unsaturated fitted models, the difference between their individual LRT statistics in comparison with the saturated model has an approximate chi-squared null distribution, and this approximation is better than the chi-squared approximation by each LRT statistic alone.

Overall chi-squared goodness-of-fit tests for a logistic regression model can only be done for categorical predictors, and for continuous predictors only if they are categorized. This is because the number of unique predictor combinations grows with increasing sample size when the predictors remain continuous.

When a logit model includes categorical predictors (factors), there is a parameter for each category of each predictor. However, one of those parameters per predictor is redundant. Setting the parameter for the last category equal to zero, and changing the definition of the remaining parameters to that of deviation from this last parameter, will eliminate redundancy of parameters in the logit model. But, the interpretation of each parameter is modified. Another way to eliminate redundancy is to set "sum-to-zero constraints", where the sum of the parameters for a particular factor is constrained to equal zero. However, any constraints for the category parameters does not affect the meaning of estimates of the odds ratios between two categories or of the joint probabilities.

Multiple logistic regression is the direct extension of univariate logistic regression. The multiple logistic regression model is given in equations (5.8) and (5.9) in Agresti. In that representation, the quantity $\exp(\beta_i)$ for the ith covariate represents the multiplicative effect on the odds of a 1-unit increase in that covariate, at fixed levels of the other covariates, provided there are no interactions between the ith covariate and other covariates. With all categorical predictors, the model is called a logit model. A

representation of a logit model with two predictors is given on p. 184 of Agresti. A logit model that does not contain an interaction between two categorical predictors assumes that the odds ratio(s) between one of the predictors and the binary dependent variable is(are) the same at or given each level of the other predictor. (In the case of a binary predictor, there is just one odds ratio between that predictor and the dependent variable). Then, the predictor and the dependent variable are said to be conditionally independent. In terms of the model representation, this means that the parameters corresponding to the categories of that predictor are all equal (see p. 184 in Agresti). A formal test of homogeneous odds ratios can be carried out using the chi-squared goodness-of-fit statistic G^2.

It is possible for additivity of predictors (i.e., no interactions) to hold on the logit scale, but not other link scales or vice versa.

B. Logistic Regression for Horseshoe Crab Data

One interpretation of the horseshoe crab data of Table 4.3 in Agresti has a binary response with a positive response being that the female crab has at least one satellite. So, a logistic regression is plausible for describing the relationship between width of carapace and probability of at least one satellite. If the widths are grouped into eight categories (Table 4.4, p. 129 Agresti), then a plot of the means of the width categories by the proportion of female crabs within each category having satellites is in Figure 5.2 of Agresti, with a logistic regression fit superimposed.

We plotted these proportions in Section I of Chapter 4, where we superimposed a GAM with logit link and binary response. Now, we will superimpose a logistic regression function.

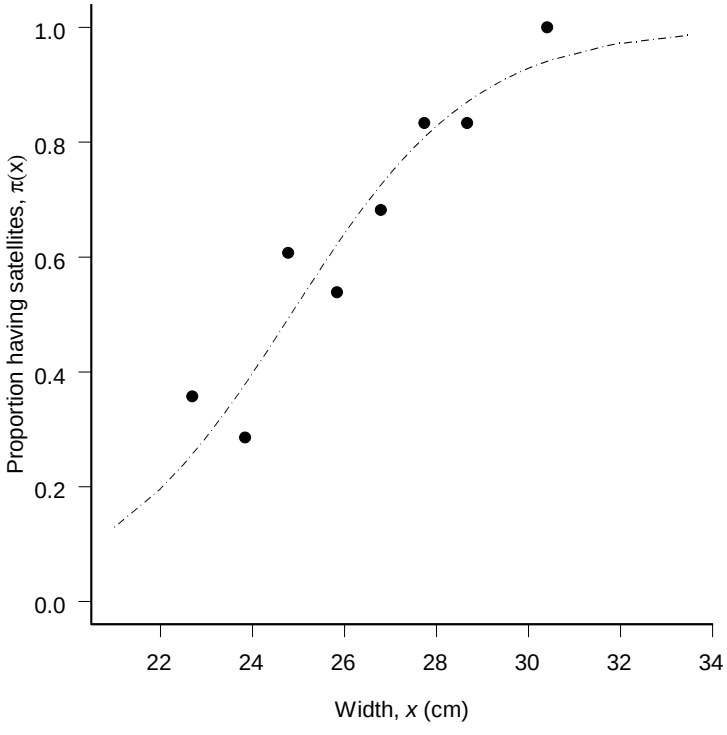
Here is the logit fit, using `glm`.

```
table.4.3$Sa.bin<-ifelse(table.4.3$Sa>0,1,0)   # change number of satellites to binary
   response
(crab.fit.logit<-glm(Sa.bin~W, family=binomial, data=table.4.3))

Call:
glm(formula = Sa.bin ~ W, family = binomial, data = table.4.3)

Coefficients:
 (Intercept)         W
   -12.35082 0.4972305

Degrees of Freedom: 173 Total; 171 Residual
Residual Deviance: 194.4527
```

So, the estimated odds of having a satellite increase by 1.64 for each 1 cm increase in width (a 64% increase). Figure 5.2 is created using similar steps as before, except now we add the predicted logistic regression curve.

```
# Recall from above the definitions:
# table.4.3$W.fac<-cut(table.4.3$W, breaks=c(0,seq(23.25, 29.25),Inf))
# prop<-aggregate(table.4.3$Sa.bin, by=table.4.3$W.fac, mean)$x
# R: prop<-aggregate(table.4.3$Sa.bin, by=list(W=table.4.3$W.fac), mean)$x
# plot.x<-aggregate(table.4.3$W, by=list(W=table.4.3$W.fac), mean)$x

old.par<-par(pty="s")   # save previous pty setting (use par(old.par) later to reset
   previous setting)
par(pty="s")            # change pty par setting to "s"

# create axes and labels
plot(y=table.4.3$Sa.bin,x=table.4.3$W,xlab="", ylab="", axes=F, type="n")
axis(2, at=seq(0,1,.2))
axis(1, at=seq(20,34,2))
guiModify( "XAxisTitle", Name = "GSD2$1$Axis2dX1$XAxisTitle",  Title = "Width, `x`
   (cm)")    # S-PLUS only
guiModify( "YAxisTitle", Name = "GSD2$1$Axis2dY1$YAxisTitle",  Title =  "Proportion
   having satellites, \\\"Symbol\"p(\\\"Arial\"x\\\"Symbol\")") # S-PLUS only
```

```
# plot points and regression curve (note the ordering of the widths first)
lines(y=prop, x=plot.x, pch=16, type="p")
ind<-order(table.4.3$W)
lines(x=table.4.3$W[ind],y=predict(crab.fit.logit, type="response")[ind], type="l",
   lty=3)
par(old.par)            # change back to old pty setting
```



For the plot in R, use

```
plot(y=table.4.3$Sa.bin,x=table.4.3$W,xlab=expression(paste("Width, ", italic(x),
"(cm)")), ylab=expression(paste("Proportion having satellites,", {pi}, "(x)")),
axes=F, type="n")
```

and do not use the two `guiModify` lines.

Inference for the logistic regression is asymptotic. Standard errors via the inverse of observed Fisher information can be obtained (among other ways) using the `summary.glm` function.

```
summary(crab.fit.logit, correlation=F)

Call: glm(formula = Sa.bin ~ W, family = binomial, data = table.4.3)

Coefficients:
                Value Std. Error   t value
(Intercept) -12.3508154  2.6280373 -4.699635
          W   0.4972305  0.1017079  4.888809

(Dispersion Parameter for Binomial family taken to be 1 )

    Null Deviance: 225.7585 on 172 degrees of freedom

Residual Deviance: 194.4527 on 171 degrees of freedom

Number of Fisher Scoring Iterations: 4
```

The Wald test is shown in the `t value` column under `Coefficients` (in R, this is a `z value`). To get the LRT, we need the log likelihood value at the estimate and at the null value 0. For this we can use the deviance values.

```
crab.fit.logit$null.deviance-crab.fit.logit$deviance
[1] 31.30586
```

A profile likelihood ratio confidence interval can be found easily in R using package `Bhat`. We first define the negative log likelihood, then use `plkhci` in the same way as in Chapter 2.

```
library(Bhat)

# neg. log-likelihood of logistic model with width included
nlogf <- function (p) {
    alpha<-p[1]; beta<-p[2]
    y<-table.4.3$Sa.bin
    lp<-alpha+beta*table.4.3$W

    -sum(y*lp - y*log(1+exp(lp)) - (1-y)*log(1+exp(lp)))

    }

# define a list with parameter labels and estimates
x <- list(label=c("alpha", "beta"),est=c(-12.3508154, 0.4972305),low=c(-100,-100),
upp=c(100,100))  # we include upper and lower bounds for stability

# CI on beta
plkhci(x,nlogf,"beta")

...snip
 CONVERGENCE:  4  iterations

chisquare value is:  3.823855
confidence bound of  beta  is  0.3087864
log derivatives:     5.526653
  label estimate log deriv log curv
1 alpha -7.47862 5.52665   90404.4
2 beta  0.308786 1217.3    61821100

[1] 0.3087864 0.7090134

# CI on alpha
plkhci(x,nlogf,"alpha")

...snip
 CONVERGENCE:  4  iterations

chisquare value is:  3.828054
confidence bound of  alpha  is  -17.79965
log derivatives:     -24.78501
  label estimate log deriv log curv
1 alpha -17.7997 30.2999   68230.4
2 beta  0.708216 -24.785   48167900

[1] -17.799654  -7.467987
```

Thus, the confidence intervals match those obtained by SAS, appearing in Table 5.1 in Agresti. A 1-cm increase in width has at least a 36% increase in odds (100*exp(0.308) = 136%) and at most about 100% increase (100*exp(0.709) = 203%).

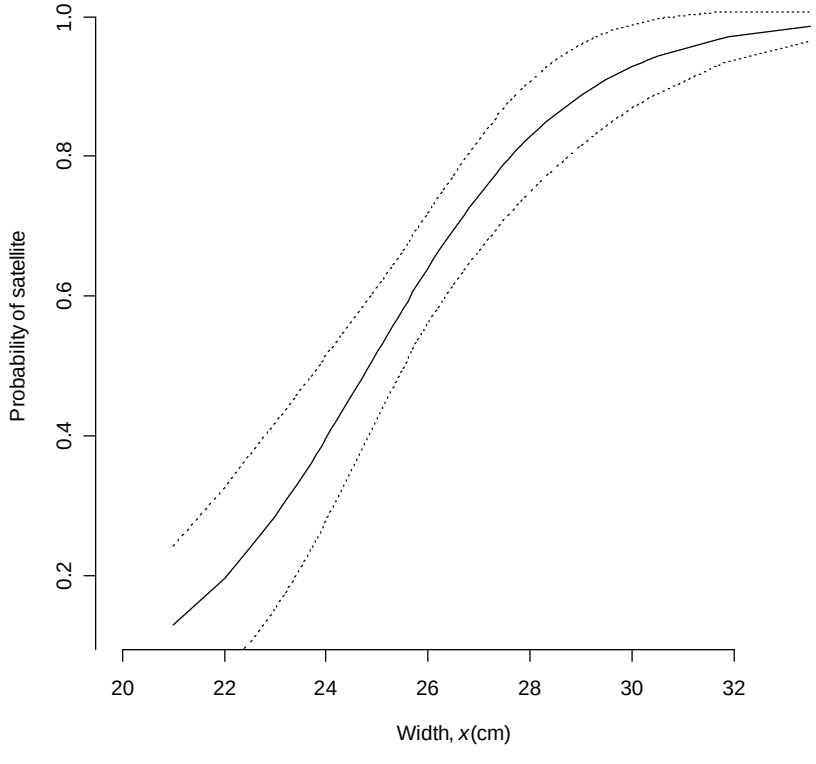
I have since found that the `MASS` library has a function `confint` that computes profile-likelihood confidence intervals for the coefficients from `glm` objects. It is much simpler to use than `plkhci` for logit models. Below, I use it on the logit model fit to the crab data

```
library(MASS)
confint(crab.fit.logit)
```

```
Waiting for profiling to be done...
                  2.5 %     97.5 %
(Intercept) -17.8104562 -7.4577421
          W   0.3084012  0.7090312
```

A plot of the predicted probabilities along with pointwise confidence intervals can be obtained using output from the `predict` function, which gives the standard errors of the predictions.

```
crab.predict<-predict(crab.fit.logit, type="response", se=T)

ind<-order(table.4.3$W)
plot(table.4.3$W[ind],crab.predict$fit[ind], axes=F, type="l", xlim=c(20,33),
   ylab="Probability of satellite", xlab="")
# R: plot(table.4.3$W[ind],crab.predict$fit[ind], axes=F, type="l", xlim=c(20,33),
   ylab="Probability of satellite", xlab=expression(paste("Width, ", italic(x),
   "(cm)")))
axis(2, at=seq(0,1,.2))
axis(1, at=seq(20,32,2))
guiModify( "XAxisTitle", Name = "GSD2$1$Axis2dX1$XAxisTitle",
   Title = "Width, `x` (cm)") # S-PLUS only
lines(table.4.3$W[ind],crab.predict$fit[ind]-1.96*crab.predict$se[ind],lty=3)
lines(table.4.3$W[ind],crab.predict$fit[ind]+1.96*crab.predict$se[ind],lty=3)
# see also the pointwise() function in S-PLUS
```



The above plot is from R. Note that it extends on the left-hand side to only a width of 21.0 cm. However, Figure 5.3 in Agresti extends to a width of 20.0 cm. As 21.0 cm is the lowest width in the data set, in order to predict the probability of a satellite at a width of 20.0 cm using the function `predict` we need to use the `newdata` argument. For example,

```
predict(crab.fit.logit, type="response", se=T, newdata=data.frame(W=seq(20,32,1)))
```

gives predictions and standard errors at widths from 20 to 32 cm, by cm.

C. Goodness-of-fit for Logistic Regression for Ungrouped Data

With categorical predictors (grouped data), one may use a chi-squared goodness-of-fit statistic where the expected number of positive and negative responses per predictor value (or predictor combination, with more than one predictor) are obtained from the fitted model. When the predictors are continuous (ungrouped data), they must be categorized prior to using the test. In that case, one may assign the midpoint of the category to the observations in that category in order to compute the expected number of positives.

In Agresti's Table 5.2, the expected number of positives in each category (Fitted Yes) is obtained by summing the predicted probabilities for each observation that falls within that category. Then, the observed number of Yes's are compared to these expected numbers in a chi-square test. Roberto Bertolusso has sent me code to compute the values within Table 5.2, and also compute the goodness-of-fit statistics. I present his code (somewhat modified) below. The original code appears in the code files for this document.

First, we create a table with the successes and failures per width category

```
cont.table<-crosstabs(~W+Sa.bin, data=table.4.3, margin=list(),drop.unused.levels=F)
#R: cont.table<-xtabs(~W+Sa.bin, data=table.4.3)
```

The unique widths can be extracted from the `dimnames` of the crosstabs.

```
w.unique <-as.numeric(attr(cont.table,"dimnames")$W)
```

This gives the observed successes and failures for each unique width, in a matrix, so that successes are listed first.

```
matrix.succ.fail<-structure(.Data=cont.table,dim=c(66,2))[,2:1]
```

Now, we create the first two columns of Table 5.2, summing elements in each column of `matrix.succ.fail` over the width categories, created using the `cut` function.

```
w.cut <- cut(w.unique, breaks=c(0,seq(23.25, 29.25),Inf), left.include=T)
observed<-apply(matrix.succ.fail,2,aggregate,by=list(W=w.cut),sum)
observed <- matrix(c(observed[[1]][,ncol(observed.yes)],
   observed[[2]][,ncol(observed.no)]), ncol = 2)
```

The last two columns contain the expected numbers of observations per category. The expected number for each category is obtained by multiplying the fitted probability for each width in the category by the number of observations at that width, and then summing up all these quantities.

```
fit.1ogit <- glm(matrix.succ.fail~w.unique, family=binomial)
fitted.yes <- aggregate(predict(fit.1ogit, type="response") *
   apply(matrix.succ.fail,1,sum), by=list(W=w.cut), sum)
fitted.no <- aggregate((1-predict(fit.1ogit, type="response")) *
   apply(matrix.succ.fail,1,sum), by=list(W=w.cut), sum)
fitted.all <- matrix(c(fitted.yes$x,fitted.no$x), ncol = 2)
```

The Pearson chi-squared statistic is then easily computed

```
(x.squared = sum((observed - fitted.all)^2/fitted.all))
[1] 5.320099

df <- length(observed[,1]) - length(fit.logit$coefficients)
1-pchisq(x.squared, df)
[1] 0.2560013
```

The likelihood ratio statistic is computed using the midpoints of the category spreads.

```
glm(observed ~ seq(22.75, 29.75), family = binomial)$deviance
[1] 6.24532
```

However, one might also consider taking the medians of the categories, as suggested by Roberto.

```
W.fac<-cut(table.4.3$W, breaks=c(0,seq(23.25, 29.25),Inf),left.include=T)
glm(observed~aggregate(table.4.3$W, by=list(W=W.fac), median)$x,
    family=binomial)$deviance
[1] 6.03537
```

which gives a slightly lower deviance.

Instead of categorizing predictors one may use a test by Hosmer and Lemeshow, described by Agresti. Their statistic forms groups based on the predicted probabilities. The observed counts per group and the predicted probabilities are used in a Pearson-like statistic that has an approximate chi-squared distribution if the number of distinct patterns of covariate values equals the sample size. It is computed here for the horseshoe crab data. First, I create a grouping variable that groups the predicted probabilities into ten groups. I give two alternative ways to do the grouping. Then, I calculate the statistic for each group using the `by()` function. The value of the statistic differs somewhat from the number that Agresti gives on p. 179. This may be due to the difference in forming groups.

```
table.4.3$prob.group<-cut(crab.predict$fit,breaks=quantile(crab.predict$fit,
    seq(0,1,.1)), include.lowest=T )
#table.4.3$prob.group<-cut(crab.predict$fit,breaks=10)
#table.4.3$prob.group<-cut(order(crab.predict$fit), breaks=seq(0,173,17.3),
    include.lowest=T)
table.4.3$predict<-crab.predict$fit

Hosmer.GOF<-sum(unlist(by(table.4.3, table.4.3$prob.group, function(x){
    p<-sum(x$predict)
    ((sum(x$Sa.bin)-p)^2)/(p*(1-p/nrow(x)))
})))

[1] 4.38554

1-pchisq(Hosmer.GOF,df=8)
# R: pchisq(Hosmer.GOF,df=8,lower.tail=F)

[1] 0.8207754
```

D. Logit Models with Categorical Predictors

As discussed in Agresti, when fitting logit models with categorical predictors, we have to constrain the category parameters to avoid redundancy in the model specification. We can set up either of the two types of constraints mentioned. The constraints are set in S-PLUS and R by specifying a global option via the `options` command. For example, to set “sum-to-zero” constraints, use

```
options(contrasts=c("contr.sum", "contr.poly"))
```

To constrain the first category parameter to be zero, use

```
options(contrasts=c("contr.treatment", "contr.poly"))
```

Thus, to fit a logit model to the data in Table 5.3 on maternal alcohol consumption and child's congenital malformations, we use `glm` with `options` set according to the constraint used.

```
Alcohol<-factor(c("0","<1","1-2","3-5",">=6"), levels=c("0","<1","1-2","3-5",">=6"))
malformed<-c(48,38,5,1,1)
n<-c(17066,14464,788,126,37)+malformed
```

To set the first category parameter to zero,

```
options(contrasts=c("contr.treatment", "contr.poly"))
(Table.5.3.logit<-glm(malformed/n~Alcohol,family=binomial, weights=n)) # saturated
   model

Coefficients:
 (Intercept)   Alcohol<1 Alcohol1-2 Alcohol3-5 Alcohol>=6
   -5.873642 -0.06818947  0.8135823   1.037361   2.262725

Degrees of Freedom: 5 Total; 0 Residual
Residual Deviance: -3.394243e-012
```

To set the last category parameter to zero,

```
revAlcohol <- factor(c("0", "<1", "1-2", "3-5", ">=6"), levels = rev(c("0", "<1", "1-
   2", "3-5", ">=6")))
(Table.5.3.logit2<-glm(malformed/n~revAlcohol, family=binomial, weights = n)) #
   saturated model

Coefficients:
 (Intercept) revAlcohol3-5 revAlcohol1-2 revAlcohol<1 revAlcohol0
   -3.610918     -1.225364     -1.449142    -2.330914   -2.262725

Degrees of Freedom: 5 Total; 0 Residual
Residual Deviance: -4.331341e-012
```

Remark: There is a difference in the reported "Total" df for R and S-PLUS. S-PLUS gives the number of cells. R gives the df for a null model, that is, one with an intercept only. Thus, for the above saturated model, R gives Total (null) df = 5-1 = 4. We lose a df for estimating an intercept.

The fitted proportions are the same for each constraint. The fitted proportions are the sample proportions because each model is a saturated model that has the same number of parameters as data points (thus, 0 degrees of freedom for residual).

```
cbind(logit=predict(Table.5.3.logit), fitted.prop= predict(Table.5.3.logit, type=
   "response"))

      logit fitted.prop
1 -5.873642 0.002804721
2 -5.941832 0.002620328
3 -5.060060 0.006305170
4 -4.836282 0.007874016
5 -3.610918 0.026315789

cbind(logit=predict(Table.5.3.logit2), fitted.prop= predict(Table.5.3.logit2,type=
   "response"))

      logit fitted.prop
1 -5.873642 0.002804721
2 -5.941832 0.002620328
```

```
3 -5.060060 0.006305170
4 -4.836282 0.007874016
5 -3.610918 0.026315789
```

The sample proportions tend to increase with alcohol consumption.

A model that specifies independence between alcohol consumption and congenital malformations is fit by

```
(Table.5.3.logit3 <- glm(malformed/n~1, family=binomial, weights = n))

Coefficients:
 (Intercept)
   -5.855811

Degrees of Freedom: 5 Total; 4 Residual
Residual Deviance: 6.201998
```

with likelihood-ratio and Pearson chi-squared statistics

```
# LR statistic
summary(Table.5.3.logit3)$deviance
[1] 6.201998

# Pearson chi-squared statistic
sum(residuals(Table.5.3.logit3, type="pearson")^2)
[1] 12.08205
```

The latter rejects the hypothesis of model fit.

```
1-pchisq(12.08205, df=4)
[1] 0.01675144
```

1. Linear Logit Model

As mentioned by Agresti, these statistics ignore ordinality in the levels of alcohol consumption. A logit model that incorporates monotone ordered categories of a predictor, but is more parsimonious than a saturated model, is a *linear logit model*. This models the logit for the *i*th category as in equation (5.5) in Agresti. To fit the model, one needs numerical scores to represent the ordered categories. For the congenital malformation data, Agresti uses scores {0, 0.5, 1.5, 4.0, 7.0} for the predictor alcohol consumption. In S, this model can be fit using a numeric vector representing the scores.

```
scores<-c(0,.5,1.5,4,7)
Table.5.3.LL<-glm(malformed/n~scores,family=binomial,weights=n)
summary(Table.5.3.LL)

(.. snip)

Coefficients:
                Value Std. Error    t value
(Intercept) -5.9604602  0.1153620 -51.667434
     scores  0.3165602  0.1254448   2.523503

(Dispersion Parameter for Binomial family taken to be 1 )

    Null Deviance: 6.201998 on 4 degrees of freedom

Residual Deviance: 1.948721 on 3 degrees of freedom

Number of Fisher Scoring Iterations: 8

Correlation of Coefficients:
```

```
       (Intercept)
scores -0.436482

# chi-squared statistic
sum(residuals(Table.5.3.LL, type="pearson")^2)
[1] 2.050051

# LR statistic
Table.5.3.LL$null.deviance - Table.5.3.LL$deviance
[1] 4.253277
```

with fitted logits and proportions

```
cbind(logit = predict(Table.5.3.LL), fitted.prop = predict(Table.5.3.LL, type =
   "response"))

      logit fitted.prop
1 -5.960460 0.002572092
2 -5.802180 0.003011863
3 -5.485620 0.004128846
4 -4.694219 0.009065079
5 -3.744539 0.023100295
```

Profile likelihood confidence intervals can be obtained using the `plkhci` function in the R package `Bhat`, which is illustrated in subsection B of this chapter, or more easily using the `confint` function from library `MASS`.

```
library(MASS)
confint(Table.5.3.LL)

Waiting for profiling to be done...
                 2.5 %     97.5 %
(Intercept) -6.19303606 -5.7396909
     scores  0.01865425  0.5236161
```

A logit model with an ordered categorical predictor can also be fit using orthogonal polynomial contrasts. However, by default, S-PLUS assumes the levels of the ordered factor are equally spaced. For illustrative purposes, you could use

```
Alcohol0<-as.ordered(Alcohol)
res<-glm(malformed/n~Alcohol0,family=binomial,weights=n)
```

to get up to quartic contrasts.

2. Cochran-Armitage Trend Test

As an alternative to the Pearson chi-squared statistic or LR statistic to test independence of alcohol on malformations, Agresti introduces the Cochran-Armitage Trend Test, which can test for a linear trend in an ordinal predictor using an I x 2 contingency table with ordered rows and I independent binomial(n_i, π_i) response variates. The test is actually equivalent to the score test for testing $H_0 : \beta = 0$ in a linear logit model: $\text{logit}(\pi_i) = \alpha + \beta x_i$. It can be calculated using the statistic M^2 in equation (3.15) in Agresti, but with $n-1$ replaced by n. Thus, for the alcohol consumption and malformation data, using the same scores as for the linear logit model, we can easily calculate the Cochran-Armitage trend statistic (denoted by z^2). We correlate the scores with the binary response variable.

```
x <- c(rep(scores, malformed), rep(scores, n - malformed))
y <- c(rep(1, sum(malformed)), rep(0, sum(n - malformed)))
(z2 <- 32574 * cor(x, y)^2)    # n = 32,574
[1] 6.569932
```

```
1 - pchisq(z2, df = 1)
[1] 0.01037159
```

which suggests strong evidence of a positive slope.

E. Multiple Logistic Regression

1. Multiple Logit Model – AIDS Symptoms Data

Agresti introduces multiple logistic regression with a data set that has two categorical predictors. Thus, the model is a logit model. Table 5.5 in Agresti cross-classifies 338 veterans infected with the AIDS virus on the two predictors Race (black, white) and (immediate) AZT Use (yes, no), and the dependent variable whether AIDS symptoms were present (yes, no). The model that is fit is the "main effects" model

$$\text{logit}\left[P(Y=1)\right] = \alpha + \beta_{yes}^{AZT} + \beta_{white}^{race}$$

In S, we will represent the predictors as factors with two levels each. This ensures that we have the correct level specifications.

```
table.5.5<-expand.grid(AZT=factor(c("Yes","No"),levels=c("No","Yes")),
   Race=factor(c("White","Black"),levels=c("Black","White")))
table.5.5<-data.frame(table.5.5,Yes=c(14,32,11,12), No=c(93,81,52,43))
```

We can fit the logit model using `glm`.

```
options(contrasts=c("contr.treatment","contr.poly"))
summary(fit<-glm(cbind(Yes,No) ~ AZT + Race , family=binomial, data=table.5.5) )

Call: glm(formula = cbind(Yes, No) ~ AZT + Race, family = binomial, data = table.5.5)

Coefficients:
                 Value Std. Error    t value
(Intercept) -1.07357363  0.2629363 -4.0830185
        AZT -0.71945990  0.2789748 -2.5789424
       Race  0.05548452  0.2886081  0.1922487

(Dispersion Parameter for Binomial family taken to be 1 )

    Null Deviance: 8.349946 on 3 degrees of freedom

Residual Deviance: 1.38353 on 1 degrees of freedom
```

Thus, the estimated odds ratio between immediate AZT use and development of AIDS is around exp(-0.7195) = 0.487. Wald confidence intervals are obtained using the approximate standard errors. LR confidence intervals can be obtained using `confint` from the MASS library.

```
confint(fit)

Waiting for profiling to be done...
                 2.5 %     97.5 %
(Intercept) -1.6088540 -0.5735061
        AZT -1.2773512 -0.1798808
       Race -0.5022939  0.6334414
```

A LRT of the conditional independence of race and AIDS symptoms, given AZT treatment is given by fitting another model which excludes race.

```
fit2<-update(object=fit, formula = ~ . -Race)

anova(fit2, fit, test="Chisq")    # R output
```

```
Analysis of Deviance Table

Model 1: cbind(Yes, No) ~ AZT
Model 2: cbind(Yes, No) ~ AZT + Race

  Resid. Df Resid. Dev Df Deviance P(>|Chi|)
1         2    1.42061
2         1    1.38353  1  0.03708   0.84730
```

Thus, the reduction in deviance (`0.03708`) is not significantly greater than chance, and we conclude that Race probably does not belong in the model.

As demonstrated above in Subsection D, one can easily change the type of constraint imposed on the parameters in order to estimate them uniquely. Use the `options` statement above in either R or S-PLUS.

The estimated probabilities for each of the four predictor combinations is given by the function `predict`, and standard errors are given by setting the argument `se` to `TRUE`.

```
res<-predict(fit, type="response", se=T)
```

To reproduce Figure 5.4 in Agresti, we need the asymptotic 95% confidence intervals on these predictions. These are just the estimate ± 1.96 times the standard error. A convenient function for getting these quantities is the function `pointwise` in S-PLUS. However, `pointwise`, by default, uses the student-t quantile as its multiplier, not the Normal multiplier. Here is a version of `pointwise` that uses a Normal quantile. I apply it to the output from `predict`.

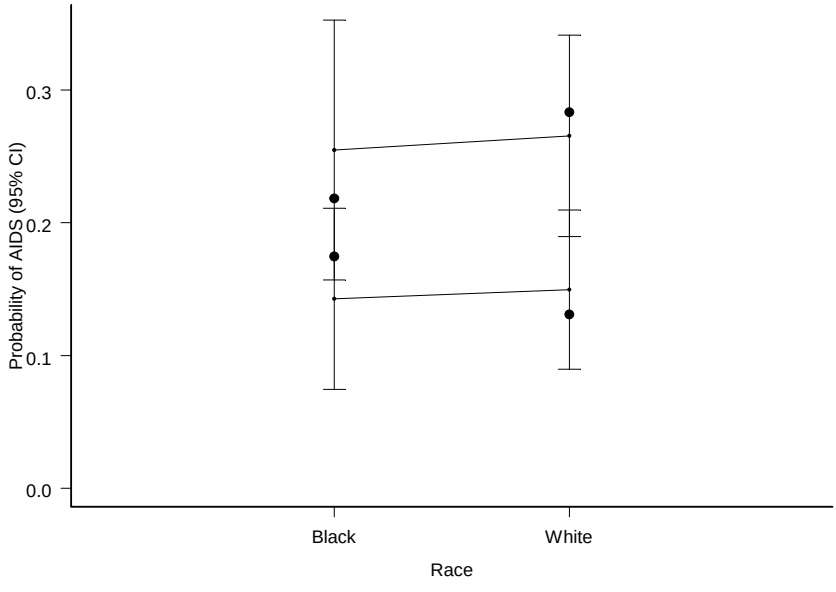
```
pointwise.normal<-function(results.predict, coverage = 0.99)
{
   fit <- results.predict$fit
   limits <- qnorm(1. - (1. - coverage)/2.) * results.predict$se.fit
   list(upper = fit + limits, fit = fit, lower = fit - limits)
}

(AIDS.bars<-pointwise.normal(res, coverage=.95))

$upper:
         1        2         3         4
 0.2095706 0.341256 0.210883 0.3525571

$fit:
         1         2         3         4
 0.1496245 0.2653998 0.1427012 0.2547241

$lower:
         1         2          3        4
 0.0896784 0.1895435 0.07451946 0.156891
```

The function `error.bar` in S-PLUS can be used to plot confidence intervals around a plotted point. For R, just `source` in the code for `error.bar` from S-PLUS (e.g., `source("c:/path/errorbar.txt")`, if the `error.bar` code is saved in the text file errorbar.txt). It works without modification (at least for this example). Here, we use it to draw a plot like Figure 5.4.

```
error.bar(c(2,2,1,1), y=AIDS.bars$fit, AIDS.bars$lower, AIDS.bars$upper,incr=F, gap=F, x
   ylab="Probability of AIDS (95% CI)", pch=".")
axis(1,at=c(2,1),labels=c("White","Black"))
axis(2,at=c(0,.1,.2,.3))

lines(c(1,2),AIDS.bars$fit[c(3,1)])
lines(c(1,2),AIDS.bars$fit[c(4,2)])
```

Now, add the sample proportions to the plot.

```
attach(table.5.5)
propAIDS<-Yes/(Yes+No)
points(c(2,2,1,1),propAIDS, pch=16)
detach(2)
```



The top line is for No AZT Use and the bottom line is for AZT Use. This plot is somewhat different-looking than that in Figure 5.4. The reason why is because Agresti has inadvertently plotted error bars which give only a single standard error on each side of the point estimate, instead of the (roughly) two standard errors from a 95% CI. Thus, the ordinate label on Figure 5.4 is incorrect. In the plot above, the confidence intervals overlap vertically.

The chi-squared and Pearson goodness-of-fit statistics for this model are obtained, respectively, by

```
fit$deviance
[1] 1.38353

sum(residuals(fit, type = "pearson")^2)
[1] 1.390965
```

which are both nonsignificant at the 0.05 level, implying that the homogeneous association model holds. Thus, the odds ratio between AZT use and AIDS symptoms is deemed to be the same regardless of race.

2. Multiple Logistic Regression Model – Horseshoe Crab Data

The Horseshoe crab data has both continuous and categorical predictors. Agresti uses the carapace width and color in a multiple logistic regression model to predict whether a crab has any satellites (the dependent variable). The predictor, color, has four categories (medium light, medium, medium dark, and dark), and it is treated in the regression model using three dummy variables. Crab color is dark when the three dummy variables are zero. Although we could set up three dummy variables, it is more natural to use `factor` in S. We construct the factor so that the coefficient set to zero is the dark color coefficient.

```
options(contrasts=c("contr.treatment", "contr.poly"))
```

```
table.4.3$C.fac<-factor(table.4.3$C, levels=c("5","4","3","2"), labels=c("dark","med-
    dark","med","med-light"))
```

MLEs of the no-interaction model can be obtained using `glm`.

```
crab.fit.logist <- glm(Sa.bin ~ C.fac + W, family = binomial, data = table.4.3)
summary(crab.fit.logist, cor = F)
```

```
Coefficients:
                    Value Std. Error   t value
   (Intercept) -12.7151039  2.7604349 -4.606196
 C.facmed-dark   1.1061211  0.5919829  1.868502
      C.facmed   1.4023356  0.5483476  2.557384
C.facmed-light   1.3299190  0.8523972  1.560210
             W   0.4679557  0.1054959  4.435769

(Dispersion Parameter for Binomial family taken to be 1 )

    Null Deviance: 225.7585 on 172 degrees of freedom

Residual Deviance: 187.457 on 168 degrees of freedom

Number of Fisher Scoring Iterations: 4
```

As before, profile likelihood confidence intervals can be obtained using the R package `Bhat`, which is illustrated in subsection B of this chapter, or using `confint` from library `MASS`.

```
library(MASS)
confint(crab.fit.logist)
```

```
                     2.5 %    97.5 %
   (Intercept) -18.45748987 -7.579268
 C.facmed-dark  -0.02793259  2.314084
      C.facmed   0.35268302  2.526314
C.facmed-light  -0.27381825  3.135721
             W   0.27129460  0.687074
```

The model has a different intercept parameter (for the linear logit) for crabs of different colors. For example, the logit model for dark crabs is $\text{logit}(\hat{\pi}) = -12.715 + 0.468\,width$; and for medium crabs it is $\text{logit}(\hat{\pi}) = (-12.715 + 1.4023) + 0.468\,width$. However, the slope on width is always the same: Regardless of color, a 1-cm increase in width has a multiplicative effect of exp(0.468) = 1.60 on the odds of having a satellite. Also, at any given width, the estimated odds that a medium crab has a satellite are exp(1.4023 – 1.1061) = 1.34 times the estimated odds for a medium-dark crab.

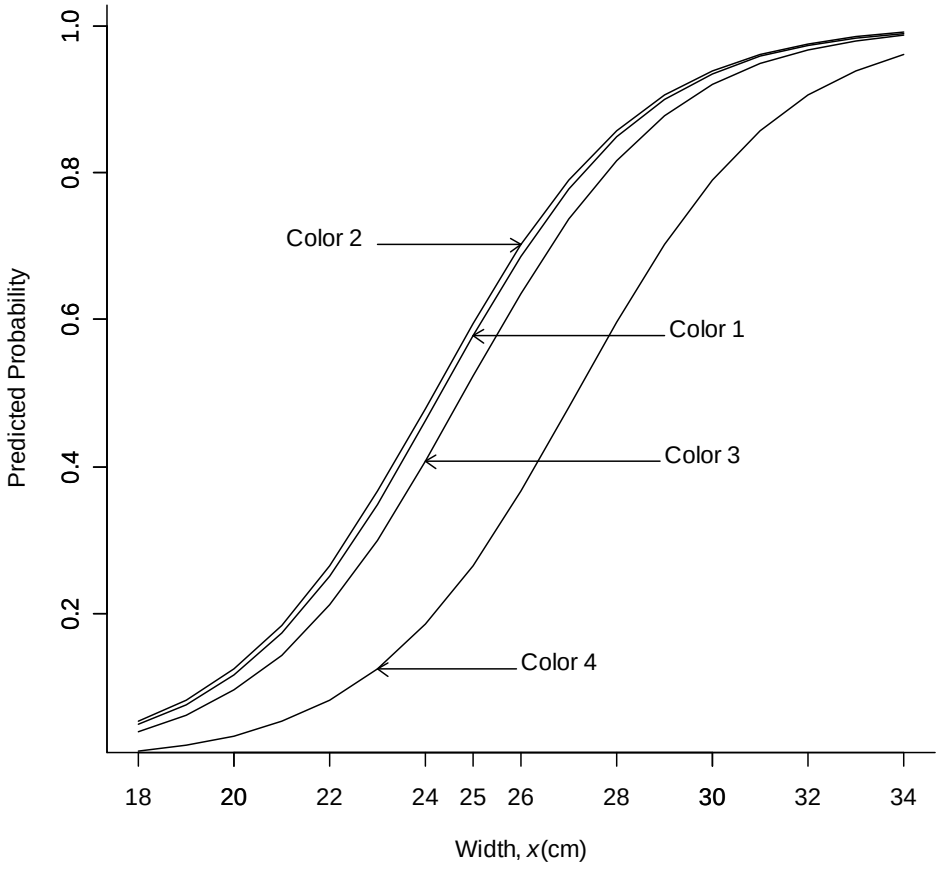
Figure 5.5 can be produced as follows. First, we predict the probability at widths from 18 to 34 cm for each of the colors.

```
res1<-predict(crab.fit.logist, type="response", newdata=data.frame(W=seq(18,34,1),
    C.fac="med-light"))
res2<-predict(crab.fit.logist, type="response", newdata=data.frame(W=seq(18,34,1),
    C.fac="med"))
res3<-predict(crab.fit.logist, type="response", newdata=data.frame(W=seq(18,34,1),
    C.fac="med-dark"))
res4<-predict(crab.fit.logist, type="response", newdata=data.frame(W=seq(18,34,1),C.fac="dark"))
```

Then, we plot the results. Here, I happen to use R.

```
plot(seq(18,34,1),res1,type="l",bty="L",ylab="Predicted Probability", axes=F,
xlab=expression(paste("Width, ", italic(x), "(cm)")))
axis(2, at=seq(0,1,.2))
axis(1, at=seq(18,34,2))
```

```
lines(seq(18,34,1),res2)   # add colors 2-4
lines(seq(18,34,1),res3)
lines(seq(18,34,1),res4)

# add arrows and text
arrows(x0=29, res1[25-17],x1=25, y1=res1[25-17], length=.09)
text(x=29.1, y=res1[25-17], "Color 1", adj=c(0,0))
arrows(x0=23, res2[26-17],x1=26, y1=res2[26-17], length=.09)
text(x=21.1, y=res2[26-17], "Color 2", adj=c(0,0))
arrows(x0=28.9, res3[24-17],x1=24, y1=res3[24-17], length=.09)
text(x=29, y=res3[24-17], "Color 3", adj=c(0,0))
arrows(x0=25.9, res4[23-17],x1=23, y1=res4[23-17], length=.09)
text(x=26, y=res4[23-17], "Color 4", adj=c(0,0))
```



As mentioned previously, there are some differences in the use of `arrows` and `text` across R and S-PLUS. Here is the code to plot Figure 5.5 in S-PLUS.

```
plot(seq(18,34,1),res1,type="l",bty="L",ylab="Predicted Probability",axes=F)
axis(2, at=seq(0,1,.2))
axis(1, at=seq(18,34,2))
guiModify( "XAxisTitle", Name = "GSD2$1$Axis2dX1$XAxisTitle",  Title= "Width,`x`
    (cm)")
lines(seq(18,34,1),res2)
lines(seq(18,34,1),res3)
lines(seq(18,34,1),res4)
arrows(x1=29, y1=res1[25-17],x2=25, y2=res1[25-17],size=.25,open=T)
text(x=29.1, y=res1[25-17], "Color 1", adj=0)
```

```
arrows(x1=23, y1=res2[26-17],x2=26, y2=res2[26-17], size=.25,open=T)
text(x=21.1, y=res2[26-17], "Color 2", adj=0)
arrows(x1=28.9, y1=res3[24-17],x2=24, y2=res3[24-17], size=.25,open=T)
text(x=29, y=res3[24-17], "Color 3", adj=0)
arrows(x1=25.9, y1=res4[23-17],x2=23, y2=res4[23-17], size=.25,open=T)
text(x=26, y=res4[23-17], "Color 4", adj=0)
```

To test whether the width effect changes at each color, we can test the significance of a color by width interaction by fitting a new model with the addition of this interaction and comparing the model deviance with that of the previous fit.

```
crab.fit.logist.ia <- update(object = crab.fit.logist, formula =  ~ . + W:C.fac)
anova(crab.fit.logist, crab.fit.logist.ia, test = "Chisq")

Analysis of Deviance Table

Response: Sa.bin

                 Terms Resid. Df Resid. Dev     Test Df Deviance   Pr(Chi)
1           C.fac + W       168   187.4570
2 C.fac + W + W:C.fac       165   183.0806 +W:C.fac  3 4.376405 0.2235832
```

The p-value implies that an interaction model is not warranted.

3. Multiple Logistic Regression Model with Quantitative Ordinal Predictor – Horseshoe Crab Data

The predictor color is actually ordinal. Agresti uses codes of {1, 2, 3, 4} for the four levels of color and fits a linear effect of color on the log odds of having a satellite. This model is easily fit using the variables in the data frame `table.4.3`, as C is already coded consecutively.

```
crab.fit.logist.ord<-glm(Sa.bin~C+W, family=binomial, data=table.4.3)
summary(crab.fit.logist.ord, cor=F)

Coefficients:
               Value Std. Error   t value
(Intercept) -9.5617875  2.8819273 -3.317845
          C -0.5090466  0.2236485 -2.276101
          W  0.4583095  0.1039784  4.407737

(Dispersion Parameter for Binomial family taken to be 1 )

    Null Deviance: 225.7585 on 172 degrees of freedom

Residual Deviance: 189.1212 on 170 degrees of freedom
```

To test the hypothesis that the quantitative color model is adequate given that the qualitative color model holds, we can use `anova`.

```
anova(crab.fit.logist.ord,crab.fit.logist, test="Chisq") # S-PLUS output

Analysis of Deviance Table

Response: Sa.bin

Response: Sa.bin

      Terms Resid. Df Resid. Dev    Test Df Deviance   Pr(Chi)
1     C + W       170   189.1212
2 C.fac + W       168   187.4570 1 vs. 2  2 1.664145 0.4351466
```

Judging by the p-value, we can go with the simpler (fewer parameters) quantitative color model.

Other scores can be created by using logical operators. For example, the set of binary scores {1, 1, 1, 0} are created by

```
table.4.3$C.bin<-ifelse(table.4.3$C<5,1,0)
glm(Sa.bin~C.bin+W, family=binomial, data=table.4.3)

Coefficients:
 (Intercept)    C.bin        W
   -12.97953 1.300512 0.478222

Degrees of Freedom: 173 Total; 170 Residual
Residual Deviance: 187.9579
```

F. Extended Example (Problem 5.17)

This example illustrates some details in using S for a logit model. The analysis is patterned after the section on binomial data in Venables and Ripley (2002, p. 190). Problem 5.17 in Agresti (p. 204) describes data on 35 patients who received general anesthesia for surgery. The dependent variable is whether the patient experienced a sore throat upon awakening (binary response). Here, we model the probability of sore throat as a logistic function of duration of surgery in minutes and the type of device used to secure the airway (0 = laryngeal mask airway, 1 = tracheal tube).

First, we set the type of contrast to treatment contrasts for factors.

```
options(contrasts=c("contr.treatment","contr.poly"))
```

Now, we get the data set up:

```
duration<-c(45,15,40,83,90,25,35,65,95,35,75,45,50,75,30,25,20,60,70,30,60,
   61,65,15,20,45,15,25,15,30,40,15,135,20,40)
type<-c(0,0,0,1,1,1,rep(0,5),1,1,1,0,0,1,1,1,rep(0,4),1,1,0,1,0,1,0,0,rep(1,4))
sore<-c(0,0,rep(1,10),0,1,0,1,0,rep(1,4),0,1,0,0,1,0,1,0,1,1,0,1,0,0)

sore.fr<-cbind(duration, type, sore)
```

Now, fit a binomial glm with interaction:

```
sorethroat.lg<-glm(sore ~ type*duration, family=binomial)
summary(sorethroat.lg, cor=T)

Coefficients:
                  Value Std. Error     t value
  (Intercept)  0.04978674 1.46940067  0.03388234
         type -4.47205400 2.45694142 -1.82017120
     duration  0.02847802 0.03428574  0.83060812
type:duration  0.07459608 0.05748718  1.29761230

(Dispersion Parameter for Binomial family taken to be 1 )

    Null Deviance: 46.17981 on 34 degrees of freedom

Residual Deviance: 28.32105 on 31 degrees of freedom

Number of Fisher Scoring Iterations: 5

Correlation of Coefficients:
             (Intercept)       type   duration
```

```
         type -0.5980609
     duration -0.9190218   0.5496310
type:duration  0.5481108  -0.9137683 -0.5964068
```

The high negative correlation between duration and the intercept can probably be reduced by standardizing duration:

```
sorethroat.lg<-glm(sore ~ type*scale(duration), family=binomial)
summary(sorethroat.lg)

Coefficients:
                         Value Std. Error    t value
         (Intercept)  1.3589619  0.6216850  2.1859332
                type -1.0427657  1.0744104 -0.9705470
     scale(duration)  0.7953023  0.9574941  0.8306081
type:scale(duration)  2.0832361  1.6054380  1.2976123

(Dispersion Parameter for Binomial family taken to be 1 )

    Null Deviance: 46.17981 on 34 degrees of freedom

Residual Deviance: 28.32105 on 31 degrees of freedom

Number of Fisher Scoring Iterations: 5

Correlation of Coefficients:
                     (Intercept)       type scale(duration)
                type -0.5786290
     scale(duration)  0.3631314  -0.2101184
type:scale(duration) -0.2165740   0.3701497 -0.5964068
```

The interaction does not appear significant based on the Wald test. A LRT of the interaction parameter gives

```
sorethroat.lg2<-glm(sore ~ type + scale(duration), family=binomial)# no interaction
anova(sorethroat.lg2, sorethroat.lg, test = "Chisq")

Analysis of Deviance Table

Response: sore

                  Terms Resid. Df Resid. Dev                 Test Df Deviance   Pr(Chi)
1 type + scale(duration)        32   30.13794
2 type * scale(duration)        31   28.32105 +type:scale(duration)  1 1.816886 0.1776844
```

which indicates that an interaction may not really be present.

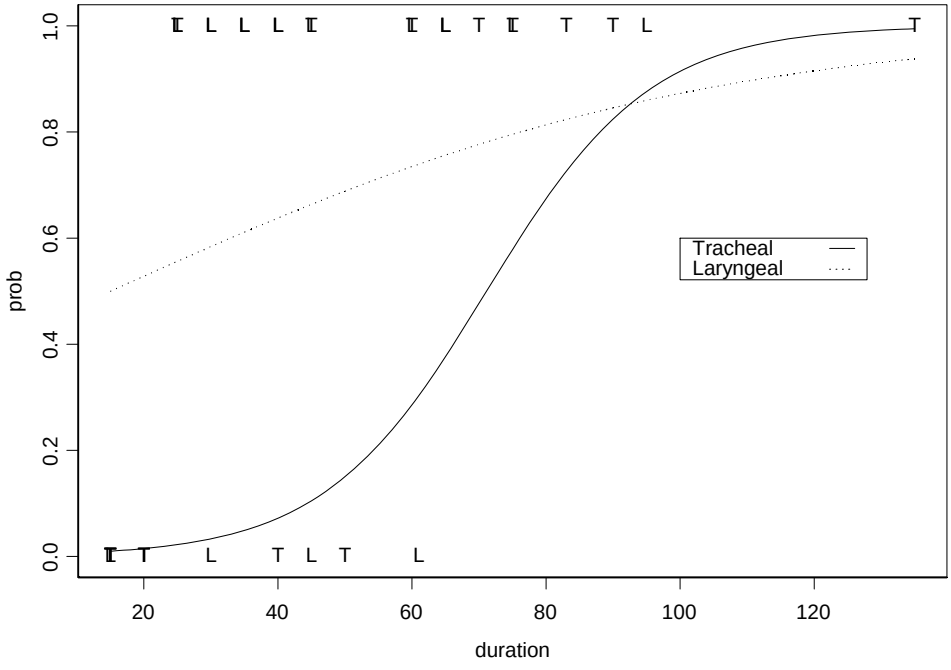
We can plot the predicted probabilities. First, we plot the data using “T” and “L” to indicate tracheal tube or laryngeal mask, respectively.

```
plot(c(15,135),c(0,1), type="n", xlab="duration",ylab="prob")
text(duration,sore,as.character(ifelse(type,"T","L")))
```

Now, we add the predicted lines for tracheal tube and laryngeal mask

```
lines(15:135,predict.glm(sorethroat.lg,data.frame(duration=15:135,type=1),
   type="response"))
lines(15:135,predict.glm(sorethroat.lg,data.frame(duration=15:135,type=0),
   type="response"), lty=2)
key(x=100, y=.6, text=list(c("Tracheal","Laryngeal")), lines=list(lty=1:2, size=2),
   border=T) # S-PLUS only
```

```
# R: legend(x=100, y=.6, legend=list("Tracheal","Laryngeal"), lty=1:2)
```



We can test for a `type` difference at a particular `duration`, say at 60 minutes, which is about 0.5 standard deviations based on the mean and standard deviation of duration. The `I( )` function is used so that (scale(duration)-0.5) is interpreted *as is*, meaning as a number, here.

```
sorethroat.lgA <- glm(sore ~ type * I(scale(duration) - 0.502), family = binomial)
summary(sorethroat.lgA) # S-PLUS output

Coefficients:
                                    Value Std. Error     t value
                      (Intercept) 1.758203676  0.9135373 1.924610748
                             type 0.003018773  1.5636194 0.001930632
     I(scale(duration) - 0.502) 0.795302340  0.9574941 0.830608118
type:I(scale(duration) - 0.502) 2.083236102  1.6054380 1.297612304

(Dispersion Parameter for Binomial family taken to be 1 )

    Null Deviance: 46.17981 on 34 degrees of freedom

Residual Deviance: 28.32105 on 31 degrees of freedom

Number of Fisher Scoring Iterations: 5
```

Based on the very low magnitude t-value, there is no difference at one hour of duration of surgery.

Now, we test a model with parallel lines for each type (common slope), but different intercepts. The term '-1' in the update formula removes the common intercept (forces it through 0) and gives separate intercepts across types.

```
type.fac <- factor(ifelse(type, "trach", "laryn"), levels = c("trach", "laryn"))
sorethroat.lgB <- update(sorethroat.lg, . ~ type.fac + scale(duration) - 1)
summary(sorethroat.lgB)$coefficients

                     Value Std. Error   t value
  type.factrach 0.08092534  0.6909792 0.1171169
  type.faclaryn 1.73987442  0.6900319 2.5214407
scale(duration) 1.91795638  0.7373841 2.6010274
```

One result we can obtain from setting different intercepts is a prediction of the duration at which prob(sore throat upon awakening) = .25, .5, .75 for tracheal tubes/laryngeal mask. We can use the function `dose.p` from the MASS library. The second argument to the function (i.e., c(1, 3) or c(2, 3)) refers to the coefficients specifying the common slope and separate intercept. For tracheal tubes, they are the `type.factrach` coefficient and `scale(duration)` coefficient. The third argument refers to the probability points (.25, .5, .75).

For tracheal tubes, we have the following predictions:

```
library(MASS)
dose.p(sorethroat.lgB, c(1, 3), (1:3)/4)
                 Dose        SE
p = 0.25: -0.61499711 0.3808615
p = 0.50: -0.04219353 0.3567465
p = 0.75:  0.53061006 0.4543964
```

And, for laryngeal mask, we have:

```
dose.p(sorethroat.lgB, c(2, 3), (1:3)/4)
                Dose        SE
p = 0.25: -1.4799537 0.5190731
p = 0.50: -0.9071502 0.3720825
p = 0.75: -0.3343466 0.3231864
```

Of course, the predicted durations are in standard deviation units. The probability of sore throat is predicted to be higher at lower durations (“doses”) when one is using the laryngeal mask.

To print residual plots for this model, use the `plot.glm` function, which is called via `plot` with first argument a `glm` object.

```
par(mfrow=c(2,2))
plot(sorethroat.lgB,ask=T)
```

Chapter 6 – Building and Applying Logistic Regression Models

A. Summary of Chapter 6, Agresti

This chapter discusses logistic regression further, emphasizing practical issues related to model choice and model assessment. In Section 6.6, other link functions besides logit link are discussed for binary data.

We would like to fit a model that is rich enough to describe the data, but does not overfit the data. We also must be aware of issues such as multicollinearity among predictors. Multicollinearity may cause related predictors to appear nonsignificant marginally. Stepwise procedures (forward selection, backward elimination, or both) can be used to select predictors for an exploratory analysis. Forward selection adds terms sequentially until further terms do not improve the fit (based on a criterion such as Akaike's Information Criterion, AIC). Backward elimination starts with a model containing many terms and sequentially removes terms that do not add "significantly" to the fit. Once the final model is obtained, p-values must be interpreted cautiously because they are usually based on knowing the model form prior to looking at the data (whereas, with stepwise selection we use the data to choose the model). Bootstrap adjustments may be needed for hypothesis tests and standard errors. In certain cases, conceptually important predictors should be included in a model, even though they may not be statistically significant.

Model building can take advantage of causal diagrams that dictate conditional independence relations among a set of variables. In this way, the causal diagram guides what models should be fit. Agresti gives an example from a British data set on extra-marital sex.

Section 6.2 expounds on diagnostics for logistic regression analysis. These include residuals (Pearson and deviance), influence diagnostics or case-deletion diagnostics (e.g., Dfbetas, and confidence interval diagnostic that measures the change in a joint confidence region on a set of parameters after deleting each observation), and measures of predictive power (e.g. R-squared like measures, ROC curves).

Section 6.3 deals with conditional inference from 2 x 2 x K tables. Testing conditional independence of a binary response Y and a binary predictor X conditional on the level of a third variable Z can be done using a test of the appropriate parameter of a logit model or using the Cochran-Mantel-Haenszel (CMH) Test (The CMH test is a score statistic alternative for the LR test of the logit model parameters). The logit model tests differ depending on whether the association between the predictor and response is assumed the same at each level of the third variable (i.e., no XZ interaction) or it is assumed to differ.

The CMH test conditions on both response Y and predictor X totals within each level of Z. Then, the first cell count in each table has a hypergeometric distribution (independent of the other tables). The CMH statistic compares the sum of the first cell counts across the K tables to its expected value of conditional independence within strata. The asymptotic distribution of the statistic is chi-squared.

Section 6.4 discusses the use of parsimonious models to improve inferential power and estimation. One example of this concept is illustrated by showing the improved power in testing for an association in the presence of a logit model when a predictor has ordinal levels. In this case, the use of numerical scores in place of the nominal levels of the ordinal predictor results in fewer parameters in the model and ultimately results in better power and smaller asymptotic variability of the cell probability estimates.

Section 6.5 discusses power and sample size calculations for the two-sample binomial test, test of nonzero coefficient in logistic regression and in multiple logistic regression, and chi-squared test in contingency tables. The formulas provide rough indications of power and/or sample size, based on assumptions about the distribution of predictors and of the model probabilities expected.

When samples are small compared to the number of parameters in a logistic regression model, conditional inference may be used. With conditional inference, inference for a parameter conditions on sufficient statistics for remaining parameters, thereby eliminating them. What remains is a conditional likelihood that only depends on the parameter of interest. In many respects, a conditional likelihood can be used like an ordinary likelihood, giving conditional MLEs and asymptotic standard errors. Exact inference for parameters uses the conditional distributions of their sufficient statistics. Conditional

inference is also used for sparse tables (with many zeroes) and tables that display "separation" where the success cases all correspond to one level of the risk factor.

B. Model Selection for Horseshoe Crab Data

Backward elimination is done for the Horseshoe Crab data of Table 4.3 in order to select a parsimonious logistic regression model that predicts the probability of a female crab having a satellite. We begin by putting all the variables from Table 4.3 into a model. Agresti uses two dummy variables for the variable spline condition, which we create by forming factors on the two variables.

```
options(contrasts=c("contr.treatment", "contr.poly"))
table.4.3$C.fac<-factor(table.4.3$C, levels=c("5","4","3","2"))
table.4.3$S.fac<-factor(table.4.3$S, levels=c("3","2","1"))
```

Now, we fit the full model, with weight (Wt) being divided by 1000, as in the text. Note the use of `I()` to interpret the argument literally.

```
crab.fit.logist.full<-glm(Sa.bin~C.fac+S.fac+W+I(Wt/1000), family=binomial,
   data=table.4.3)

summary(crab.fit.logist.full, cor=T)

Coefficients:
                 Value Std. Error    t value
(Intercept) -9.2733819  3.8364636 -2.4171693
     C.fac4  1.1198011  0.5931762  1.8878052
     C.fac3  1.5057627  0.5665525  2.6577638
     C.fac2  1.6086660  0.9353686  1.7198203
     S.fac2 -0.4962679  0.6290766 -0.7888830
     S.fac1 -0.4002868  0.5025636 -0.7964899
          W  0.2631276  0.1952484  1.3476557
 I(Wt/1000)  0.8257794  0.7036640  1.1735422

(Dispersion Parameter for Binomial family taken to be 1 )

    Null Deviance: 225.7585 on 172 degrees of freedom

Residual Deviance: 185.202 on 165 degrees of freedom

Number of Fisher Scoring Iterations: 4

Correlation of Coefficients:
          (Intercept)     C.fac4     C.fac3     C.fac2     S.fac2     S.fac1          W
    C.fac4 -0.1318818
    C.fac3 -0.0670015   0.7233703
    C.fac2 -0.0043035   0.4499020  0.5507148
    S.fac2 -0.2184819  -0.0733221 -0.1685117 -0.2471148
    S.fac1 -0.0120010  -0.0327826 -0.2074473 -0.3672790  0.2431179
         W -0.9649203   0.0241011 -0.0308300 -0.0336341  0.1922667  0.0161518
I(Wt/1000)  0.6740016  -0.0097672 -0.0014684 -0.0365701 -0.0891985 -0.0402631 -0.8308544
```

with LR statistic

```
crab.fit.logist.full$null.deviance-crab.fit.logist.full$deviance
[1] 40.55652
```

Because of the high correlation between width and weight, Agresti eliminates the predictor `Wt` in further analyses.

To perform stepwise selection of predictor variables for various types of fitted models (including `glm` objects), one can use the function `step` with a `lower` and `upper` model specified. The criterion is AIC. Only the final model is printed unless `trace=T` is specified. An example of a call to the `glm` method is

```
step.glm(fit, scope=list(lower = formula(fit), upper =  ~ .^2 ), scale=1, trace=T,
   direction="both")
```

The above call specifies both forward and backward stepwise selection of terms (`direction="both"`). The `scope` of the selection has a lower bound or starting model as "`fit`". The `upper` bound model includes all two-way interactions. The component "`anova`" gives a summary of the trace path.

To illustrate stepwise procedures, we perform backward elimination on a model fitted to the horseshoe crab data. This model includes up to a three-way interaction among Color, Width, and Spine Condition. We fit this model in S-PLUS or R using

```
crab.fit.logist.stuffed<-glm(Sa.bin~C.fac*S.fac*W, family=binomial,data=table.4.3)
```

(Note that there are some warning messages after the fit.) The backward elimination begins with the above three-way interaction model. The lower bound of the `scope` is a null model. The upper bound is the saturated or "stuffed" model, as I appear to have called it.

```
res <- step.glm(crab.fit.logist.stuffed, list(lower =  ~ 1, upper =
   formula(crab.fit.logist.stuffed)), scale = 1, trace = F, direction = "backward")
res$anova

Stepwise Model Path
Analysis of Deviance Table

Initial Model:
Sa.bin ~ C.fac * S.fac * W

Final Model:
Sa.bin ~ C.fac + W


             Step Df Deviance Resid. Df Resid. Dev      AIC
1                                   152   170.4462 212.4462
2 - C.fac:S.fac:W  3 3.232082       155   173.6783 209.6783
3   - C.fac:S.fac  6 7.880494       161   181.5588 205.5588
4       - S.fac:W  2 0.078190       163   181.6370 201.6370
5         - S.fac  2 1.443615       165   183.0806 199.0806
6       - C.fac:W  3 4.376405       168   187.4570 197.4570
```

Compare the above table to Table 6.2 in Agresti. The models chosen at each step are the same as those in Table 6.2, with the exception of step 5. At step 5, `step.glm` opts to drop S instead of the two-way interaction C:W, giving a resulting AIC of 199.0806 instead of 200.6. Setting `trace=T` in the call of the function shows this explicitly. The model chosen by `step.glm` is C + W with AIC = 197.46. The iterations stop before steps 7 and 8 in Table 6.2 because at step 7, no model decreases AIC.

The `stepAIC` function from the MASS library also performs stepwise selection based on AIC. In some cases, the AIC calculation from `stepAIC` is more accurate than that of `step.glm`. See the help library for `stepAIC` or Venables and Ripley (2002). Here, `stepAIC` gives the same `anova` summary.

C. Using Causal Hypotheses to Guide Model Fitting

In this subsection, I show how to fit the various models derived from the causal diagram in Figure 6.1 in Agresti. There are no new techniques learned, but it gives an illustration of fitting several related models coming from the same contingency table. A 2x2x2x2 table of variables: gender, premarital sex,

extramarital sex, and marital status (divorced, still married) is given in Table 6.3 (p. 217, Agresti). The data come from a British survey of a sample of men and women who had petitioned for divorce, and a similar number of married people.

The causal diagram indicates a conditional independence relation: M and G are conditionally independent given E and P. Thus, if we broke the arrows connecting E to M and P to M, there would be no path between G and M. A logit model with M as response, then, might have E and P as explanatory variables, but not G. This model and the remaining in Table 6.4 (p. 218, Agresti) are fitted below using S.

First, I enter the data (see note at end of this section)

```
table.6.3<-expand.grid(list(M=c("divorced","married"),E=c("yes","no"),
   P=c("yes","no"), G=c("Female","Male")))
count<-c(17,4,54,25,36,4,214,322,28,11,60,42,17,4,68,130)
table.6.3.expand<-table.6.3[rep(1:(length(count)),count),]
```

Then, I fit the models and compare the reductions in deviance.

Stage 1:

```
EMS.10<-glm(P ~ 1, family=binomial, data=table.6.3.expand)
EMS.11<-update(EMS.10, .~. +G)
anova(EMS.10, EMS.11)

Analysis of Deviance Table

Response: P

  Terms Resid. Df Resid. Dev Test Df Deviance
1     1      1035   1123.914
2     G      1034   1048.654       1  75.2594
```

Thus, adding G reduces the deviance by about 75.3.

Stage 2:

```
EMS.20<-glm(E ~ 1, family=binomial, data=table.6.3.expand)
EMS.21<-update(EMS.20, .~. +P)
EMS.22<-update(EMS.21, .~. +G)
anova(EMS.20, EMS.21, EMS.22)

Analysis of Deviance Table

Response: E

  Terms Resid. Df Resid. Dev Test Df Deviance
1     1      1035   746.9373
2     P      1034   700.9209       1 46.01636
3 P + G      1033   698.0138   +G  1  2.90718
```

Adding P reduces the deviance by about 46 (= 48.9 – 2.9). Adding G to this reduces the deviance by 2.9 more points.

Stage 3:

```
EMS.31<-glm(M ~ E+P, family=binomial, data=table.6.3.expand)
EMS.32<-update(EMS.31, .~. +E:P)
EMS.33<-update(EMS.32, .~. +G)
anova(EMS.31, EMS.32, EMS.33)

Analysis of Deviance Table

Response: M

            Terms Resid. Df Resid. Dev Test Df Deviance
1           E + P      1033   1344.180
```

```
2     E + P + E:P      1032   1331.266 +E:P  1 12.91404
3 E + P + G + E:P      1031   1326.718   +G  1  4.54768
```

Adding E:P to a model with E and P reduces the deviance by about 13 points (=18.2 – 5.2). Further adding G reduces deviance by 4.5 points (= 5.2 – 0.7). Thus, we see how Table 6.4 is obtained.

I have since found this data set available in the R package `vcd`. Thus, if you have loaded the `vcd` library, just issue the command: `data(PreSex)`, to have the `PreSex` data array available. The function `as.data.frame()` can be used to transform it to a data frame.

D. Logistic Regression Diagnostics

This section gives more details on diagnostics for logistic regression. After illustrating each set of procedures, I use the two data sets in Subsections 6.2.2 and 6.2.3 in Agresti to demonstrate their use in S.

1. Pearson, Deviance and Standardized Residuals

We already illustrated Pearson residuals in Chapter 3, Section D.1 and Chapter 4, Section F. The sum of the squared Pearson residuals is equal to the Pearson chi-squared statistic. For a logistic regression model, with responses as counts out of totals, n_i, $i=1,\ldots,N$, the fitted response value at the ith combination of the covariates is $n_i\hat{\pi}_i$. So, the Pearson and deviance residuals use deviations of the observed responses from these fitted values. Pearson residuals are divided by an estimate of the standard deviation of an observed response, and the standardized version is further divided by the square root of the 1 – the ith estimated leverage value (ith diagonal of estimated "hat" matrix). This standardized residual is approximately distributed standard normal when the model holds. Thus, absolute values of greater than about three provide evidence of lack of fit.

Deviance residuals were illustrated for S in Chapter 4, Section F. These are the signed square roots of the components of the LR statistic. Standardized deviance residuals are approximately distributed standard normal.

These residuals can be plotted against fitted linear predictors to detect lack of fit, but as Agresti says, they have limited use. When n_i = 1, individual residuals can be either uninteresting (Pearson) or uninformative (deviance).

The heart disease data in Agresti (Table 6.5, p. 221) classifies blood pressure (BP) for a sample of male residents aged 40-59, into one of 8 categories. Then, a binary indicator response is whether each man developed coronary heart disease (CHD) during a six-year follow-up period. An independence model is fit initially. This model has BP independent of CHD.

```
BP<-factor(c("<117","117-126","127-136","137-146","147-156","157-166","167-
   186",">186"))
CHD<-c(3,17,12,16,12,8,16,8)
n<-c(156,252,284,271,139,85,99,43)
```

Independence model:

```
resCHD<-glm(CHD/n~1,family=binomial, weights=n)
resCHD$deviance
[1] 30.02257
```

Predicted responses, deviance residuals, Pearson residuals, and standardized Pearson residuals.

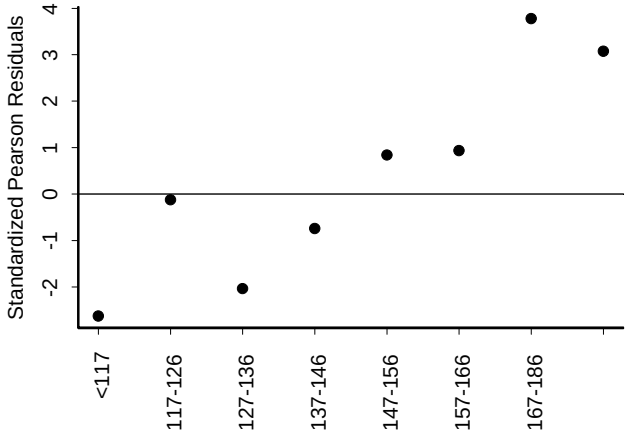
```
pred.indep<-n*predict(resCHD, type="response")
dev.indep<-resid(resCHD, type="deviance")
pear.indep<-resid(resCHD, type="pearson")
```

```
pear.std.indep<-resid(resCHD, type="pearson")/sqrt(1-lm.influence(resCHD)$hat)

structure(cbind(pred.indep, dev.indep, pear.indep, pear.std.indep), dimnames =
list(BP, c("fitted", "deviance resid",  "pearson resid", "pearson std resid")))
# R: structure(cbind(pred.indep, dev.indep, pear.indep, pear.std.indep),
    dimnames=list(as.character(BP),c("fitted","deviance resid","pearson resid",
    "pearson std resid")))
```

	fitted	deviance resid	pearson resid	pearson std resid
<117	10.799097	-0.231005277	-2.4599611	-2.6298091
117-126	17.444695	-0.006979615	-0.1103592	-0.1235339
127-136	19.659895	-0.114009224	-1.7906464	-2.0374109
137-146	18.759970	-0.041094691	-0.6604895	-0.7464933
147-156	9.622272	0.065059023	0.7945128	0.8428348
157-166	5.884123	0.093323311	0.9041221	0.9365792
167-186	6.853273	0.314268383	3.6215487	3.7743451
>186	2.976674	0.386702000	3.0178895	3.0712637

A plot of the standardized residuals shows an increasing trend.

```
plot(pear.std.indep,xlab="",ylab="Standardized Pearson Residuals", pch=16, axes=F)
axis(1,at=1:8, labels=as.character(BP), srt=90)
axis(2);abline(h=0)
```



Agresti notes that this suggests that a linear logit model may be better to use, with scores for BP.

To indicate residuals greater than | 3 | on the plot, use text and then points, as follows

```
out<-abs(pear.std.indep)>3
plot(pear.std.indep, xlab="",ylab="Standardized Pearson Residuals", axes=F, type="n")
text((1:8)[out], pear.std.indep[out], ">3")
points((1:8)[!out], pear.std.indep[!out], pch=16)
axis(1,at=1:8, labels=as.character(BP), srt=90)
axis(2)
abline(h=0)
```

Linear Logit Model:

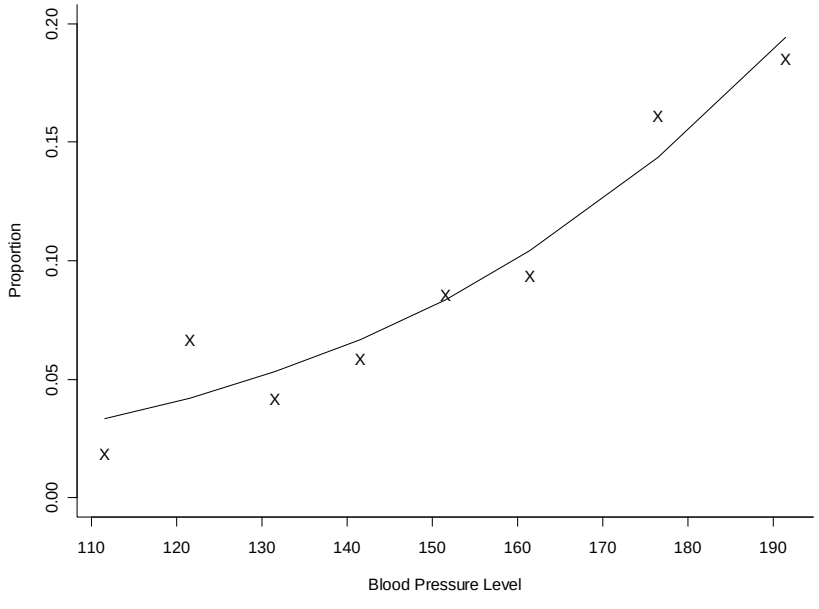
```
scores<-c(seq(from=111.5,to=161.5,by=10),176.5,191.5)
resLL<-glm(CHD/n~scores,family=binomial,weights=n)
resLL$deviance
[1] 5.909158

pred.indep<-n*predict(resLL, type="response")
```

```
dev.indep <- resid(resLL, type = "deviance")
pear.indep <- resid(resLL, type = "pearson")
pear.std.indep <- resid(resLL, type = "pearson")/sqrt(1 - lm.influence(resLL)$hat)

structure(cbind(pred.indep, dev.indep, pear.indep, pear.std.indep), dimnames =
   list(as.character(scores), c("fitted", "deviance resid", "pearson resid", "pearson
   std resid")))
```

```
         fitted deviance resid pearson resid pearson std resid
111.5  5.194869     -1.0616845    -0.9782938        -1.1046299
121.5 10.606767      1.8501055     2.0037958         2.3724980
131.5 15.072743     -0.8419675    -0.8127218        -0.9445420
141.5 18.081622     -0.5162313    -0.5064454        -0.5724088
151.5 11.616362      0.1170009     0.1175384         0.1260411
161.5  8.856988     -0.3087751    -0.3041875        -0.3260156
176.5 14.208763      0.5049658     0.5134918         0.6519597
191.5  8.361957     -0.1402427     -0.139493        -0.1773537
```

The residuals are generally smaller. We can plot the fitted proportions along with the observed proportions using the following.

```
win.graph(width=10, height=8) # R only
plot(scores,CHD/n,pch="X",yaxt="n",xaxt="n",ylim=c(0,.2),
   xlab="Blood Pressure Level",ylab="Proportion",bty="L")
axis(side=1, at=seq(from=110,to=200,by=10))
axis(side=2, at=seq(from=0,to=.2,by=.05))
lines(scores,predict(resLL, type="response"),type="l")
```



The graduate admissions data in Table 6.7 p. 223 of Agresti cross-classifies applicants on gender (G), whether admitted (A) and department of application (D). The number admitted of each gender for each department are assumed to be independent binomials with different probabilities of success.

```
yes<-c(32,21,6,3,12,34,3,4,52,5,8,6,35,30,9,11,6,15,17,4,9,21,26,25,21,7,25,31,3,9,
   10,25, 25,39,2,4,3,0,29,6,16,7,23,36,4,10)
no<-c(81,41,0,8,43,110,1,0,149,10,7,12,100,112,1,11,3,6,0,1,9,19,7,16,10,8,18,37,0,
   6,11,53,34,49,123,41,3,2,13,3,33,17,9,14,62,54)
```

```
table.6.7<-cbind(expand.grid(gender=c("female","male"),
    dept=c("anth","astr","chem","clas","comm","comp","engl","geog","geol","germ",
    "hist","lati","ling","math","phil","phys","poli","psyc","reli","roma","soci",
    "stat","zool")),prop=yes/(yes+no))
```

A model with no gender effect is

```
res.gradadmit<-glm(prop~dept, family=binomial, data=table.6.7, weights=yes+no)
res.gradadmit$deviance
[1] 44.73516
sum(resid(res.gradadmit, type="pearson")^2)
[1] 40.80606
```

Standardized Pearson residuals are obtained in the same way as above.

```
resid(res.gradadmit, type="pearson")/sqrt(1-lm.influence(res.gradadmit)$hat)
```

2. Influence Diagnostics

The estimated "hat" matrix for GLIMs can be used to assess leverage for each observation. For an observation with large leverage, if it also has an outlying residual, then deleting the observation from the model fit may cause large changes in the fit.

Case deletion diagnostics measure the change in fit after deleting an observation. A measure called Dfbeta measures the standardized change in a parameter estimate when an observation is deleted from the model fit. In S-PLUS, we can compute Dfbeta measures using the function `Cook.terms` by John Chambers (Chambers and Hastie, 1992). The signed square root of the output from this function gives the Dfbetas. Using it on the linear logit model for blood pressure, we get

```
sign(resLL$coefficients[2]-lm.influence(resLL)$coefficients[,2,drop=F])
    *sqrt(Cook.terms(resLL)[,2])
        scores
1  0.492099277
2 -1.142149031
3  0.327960528
4  0.081380326
5  0.007858333
6 -0.065196274
7  0.400949178
8 -0.123737455
```

In R, there is a function called `influence.measures()` that is used for computing influence measures for an `lm.object`. There is also a `dfbetas()` function by itself. Here, if we use `dfbetas` on the `glm.object`, `resLL`, we get values that are somewhat similar to those from S-PLUS.

```
dfbetas(resLL)[,2,drop=F]

        scores
1  0.564578573
2 -2.237161736
3  0.341496449
4  0.078670948
5  0.007193748
6 -0.061424292
7  0.375986525
8 -0.114761146
```

Influence measures for logistic regression models are also available from Harrell's library/package `Design` and `Hmisc` in the form of the function, `residuals.lrm`, with argument `type="dfbetas"`. The function takes as input the output from the Design function `lrm`. We discuss this function next.

3. Summarizing Predictive Power: R and R-squared Measures

The function `lrm` from Harrell's Design library (available for both R and S-PLUS) computes measures of predictive ability of a logistic regression model, including R and R-squared-like measures. These measures can also be validated using resampling (function `validate.lrm`). This function calculates bias-corrected estimates of predictive ability. Bias is incurred when the data used to fit a model is also used to assess its predictive ability on a potential new data set. Model selection, assessment, and tuning of a model fit are usually not taken into account when one computes a measure of predictive power. These activities can bias the computed measure upward. Thus, bias-correction is in order.

To illustrate the variety of measures computed by these functions, I use `lrm` to fit the linear logit model to the blood pressure data. `lrm` requires binary responses instead of count response, so I form `table.6.5` out of objects defined previously. Note that I set the arguments `x` and `y` to `TRUE` so that we can use `validate.lrm` later on. First, when loading the libraries, set `first=T` so that certain functions from `Design` don't get confused with built-in ones from S-PLUS (done automatically in R).

```
library(Hmisc, first=T) # first=T not needed in R
library(Design, first=T)   # R: loads Hmisc upon loading Design

#BP<-factor(c("<117","117-126","127-136","137-146","147-156","157-166","167-
   186",">186"))
#scores<-c(seq(from=111.5,to=161.5,by=10),176.5,191.5)
#CHD<-c(3,17,12,16,12,8,16,8)
#n<-c(156,252,284,271,139,85,99,43)

# form the data
res<-numeric(2*length(CHD))
res[rep(c(T,F),length(CHD))]<-CHD
res[rep(c(F,T),length(CHD))]<-n-CHD
table.6.5<-data.frame(CHD=rep(rep(c(1,0),length(CHD)),res), BP=rep(BP,n),
   scores=rep(scores,n))

# fit the linear logit model
options(contrasts=c("contr.treatment", "contr.poly"))
(res.lrm<-lrm(CHD~scores,data=table.6.5, x=T, y=T))

Logistic Regression Model

lrm(formula = CHD ~ scores, data = table.6.5, x = T, y = T)


Frequencies of Responses
    0  1
 1237 92

  Obs Max Deriv Model L.R. d.f. P     C   Dxy Gamma Tau-a    R2 Brier
 1329    2e-005      24.11    1 0 0.636 0.273 0.316 0.035 0.045 0.063

              Coef     S.E. Wald Z P
Intercept -6.08203 0.724320 -8.40  0
   scores  0.02434 0.004843  5.03  0
```

The `Model L.R.` reported by `lrm` is `resLL$null.deviance-resLL$deviance` for the same model fit using `glm` (`resLL`). Thus, it is the reduction in deviance by fitting `scores`. However, because `resLL` was fit using grouped data instead of binary responses, the absolute deviances will be different. This has implications for the reported R-squared measure. The `R2` measure reported by `lrm` is the Nagelkerke measure, R_N^2, which measures the proportionate reduction in deviance when fitting the unconstrained model versus the null model. It is computed as

$$R_N^2 = \frac{1-\exp(-(L_0 - L_M)/n)}{1-\exp(-L_0/n)}$$

where L_0 is the deviance (-2 times log likelihood) under the null model, and L_M is the deviance under the unconstrained model, and *n* is the number of responses. All three of these will differ when using grouped versus ungrouped data, and the R-squared values will differ. Consider the value of R_N^2 for the `glm` fit, `resLL<-glm(CHD/n~scores,family=binomial,weights=n)` (from p. 94):

```
(1 - exp((resLL$deviance - resLL$null.deviance)/1329))/(1 - exp( -
  resLL$null.deviance/1329))

[1] 0.8049576
```

This value is much larger. However, when we use binary responses with `glm`, we get

```
temp <- glm(formula = CHD ~ scores, family = binomial, data = table.6.5)
(1 - exp((temp$deviance - temp$null.deviance)/1329))/(1 - exp( -
  temp$null.deviance/1329))  # sum(n) = 1329

[1] 0.04546905
```

4. Summarizing Predictive Power: ROC Curve

The measure "`C`" reported by the `lrm` output above is an index of the rank correlation between the predicted probability of response under the fitted model and the actual response. It is the probability of concordance between predictions and outcomes, and is equivalent to the area under a *receiver operating characteristic* (ROC) curve. If we were to classify predicted probabilities as success if the predicted probability exceeds some fixed value π_0 and failure if not, then a plot of the sensitivity of the model (predicting success when it should) by one minus the specificity (predicting failure when it should) for a range of values of π_0 is the ROC curve. The larger the area under this curve, the better the predictions. The maximum area is 1.0, and an area of 0.5 implies random predictions (i.e., a prediction of success is as likely whether success or failure is the truth). A value of `C` = 0.636 above is not much better than random predictions, in agreement with the very low `R2` value. Harrell (1998) gives a guideline of `C` exceeding 0.80 as implying useful predictability of the model.

Somer's D is a transformation of C, equal to 2(C – 0.5), where D = 0 if predictions are random and D = 1 if predictions are perfect. The various rank measures (Somer's D, tau-a, C) only measure how well the predicted values can rank order responses, but the actual numerical predictions do not matter (i.e., only ordinal information is used).

To validate the logistic model fit, we can use `validate.lrm`, from the same library. This can be called using `validate()`, with an object of class `lrm` (but, be careful, as there is a `validate` function in S-PLUS that provides a completely different service). I will use the full name here, instead of taking advantage of method dispatch.

Below is the output of a call to `validate.lrm`, using 100 bootstrap repetitions, where sampling is done with replacement. A repetition proceeds as follows. A bootstrap sample is drawn from the original data and becomes the *training set* for the repetition. The model is fit to this sample, coefficients are estimated and measures of predictive accuracy are obtained. Another bootstrap sample is drawn (called the *test* sample), and the model is fit using the responses from that sample and one predictor: the linear predictor formed using the covariate matrix from the test sample along with the coefficient estimates from the training sample. Measures of predictive accuracy are calculated from this fit. After 100 repetitions, the training sample indexes are averaged and the test sample indexes are averaged. These averages appear in the `validate.lrm` output.

I only show the two indexes of predictive accuracy, Dxy (Somer's D) and R2, but several others are output as well. The `index.corrected` column is the row index corrected for bias due to over-optimism (i.e., `index.orig – optimism`). The `optimism` column is computed as the difference between the average of the indexes calculated from the training samples and the average calculated from the test samples. Here, there is very little over-optimism, so the corrected estimates are very close to the original indexes.

```
validate.lrm(res.lrm, method = "boot", B = 100)   # excerpt

          index.orig     training          test      optimism index.corrected   n
Dxy   0.272819936  0.270357318 0.27281993603 -0.0024626183   0.27528255435 100
R2    0.045469052  0.047315513 0.04546905186  0.0018464612   0.04362259074 100
.........snip
```

Harrell has written much about the topic of assessing predictive ability from linear, logistic, and survival models. The above information came from a preprint (Harrell, 1998) of his now-published Springer book. In addition, on the website for one of my courses at UHCL (http://math.cl.uh.edu/~thompsonla/5537), I have html documents and S-PLUS scripts illustrating many of the `lrm` model assessments on GLIMs, with annotation.

After using Design and Hmisc, we should detach them.

```
detach("Design") # R: detach("package:Hmisc")
detach("Hmisc") # R: detach("package:Design")
```

E. Inference about Conditional Associations in 2 x 2 x K Tables

In this section, I show how to test for conditional independence between response *Y* and predictor *X* in 2 x 2 x K tables using S, where K is the number of levels of a stratification variable. Agresti illustrates a test using an appropriate logit model and the Cochran-Mantel-Haeszel Test. Table 6.9 (p. 230, Agresti) gives results of a clinical trial with eight centers. The study compared two cream preparations, an active drug and a control, on their success in treating an infection. This data set is available on Agresti's text website (I have changed the name, and changed the extension to "ssc"). I read it in using `scan` in order to read the numeric levels (1 and 2) for the response and treatment as characters. Then, I make it a data frame and change the levels to something more meaningful.

```
table.6.9<-data.frame(scan(file="clinical trials table 69.ssc",
   what=list(Center="",Treatment="",Response="",Freq=0)))
levels(table.6.9$Treatment)<-c("Drug","Control")
levels(table.6.9$Response)<-c("Success","Failure")
```

The CMH test assumes that the odds ratios are in the same direction across strata. Thus, prior to conducting the test, we can check the sample odds ratios across the centers. This can be done using `oddsratio` from package `vcd` in R. The same function can be sourced into S-PLUS with modifications due to restricted variable names and scoping rule differences between R and S-PLUS. I have modified the function to be sourceable into S-PLUS as well as to use the non-corrected cell counts if desired. Here, we will use them.

```
oddsratio.L<-
function (x, stratum = NULL, Log = TRUE, conf.level = 0.95, correct=T)
{
# modified version of oddsratio in package vcd
    l <- length(dim(x))
    if (l > 2 && is.null(stratum))
        stratum <- 3:l
    if (l - length(stratum) > 2)
        stop("All but 2 dimensions must be specified as strata.")
```

```
    if (l == 2 && dim(x) != c(2, 2))
        stop("Not a 2 x 2 - table.")
    if (!is.null(stratum) && dim(x)[-stratum] != c(2, 2))
        stop("Need strata of 2 x 2 - tables.")
    lor <- function(y, correct, Log) {
        if(correct) y<-y + 0.5
        or <- y[1, 1] * y[2, 2]/y[1, 2]/y[2, 1]
        if (Log)
            log(or)
        else or
    }
    ase <- function(y, correct) sqrt(sum(1/(ifelse(correct,y + 0.5,y))))
    if (is.null(stratum)) {
        LOR <- lor(x, correct)
        ASE <- ase(x)
    }
    else {
        LOR <- apply(x, stratum, lor, correct=correct, Log=Log)
        ASE <- apply(x, stratum, ase)
    }
    I <- ASE * qnorm((1 + conf.level)/2)
    Z <- LOR/ASE
    structure(LOR, ASE = if (Log)
        ASE, lwr = if (Log)
        LOR - I
    else exp(log(LOR) - I), upr = if (Log)
        LOR + I
    else exp(log(LOR) + I), Z = if (Log)
        Z, P = if (Log)
        1 - pnorm(abs(Z)), log = Log, class = "oddsratio")
}
```

This function is meant to work in both R and S-PLUS. To use `oddsratio.L` we need the data as an array. This is accomplished using `design.table` in S-PLUS (where I've moved the columns around to match those in Table 6.9, so that Z comes after X and Y) and `xtabs` in R.

```
table.6.9.array<-design.table(table.6.9[,c(2,3,1,4)])
# R: table.6.9.array<-xtabs(Freq~Treatment+Response+Center, data=table.6.9)
```

Then, we apply the function (here, in R)

```
oddsratio.L(table.6.9.array, correct=F, Log=F)
```

```
        a         b         c         d         e         f         g         h
1.1880000 1.8181818 4.8000000 2.2857143       Inf       Inf 2.0000000 0.3333333
```

If the odds ratios are very different across strata, we might opt to test for conditional independence by comparing a logit model with no *X* effect to a saturated model with *X* effect, *Z* effect and *XZ* association. If we assume that the actual odds ratios are nearer the same, we can use the Cochran-Mantel-Haenszel test, with asymptotic null chi-squared distribution. It is available in both S-PLUS and R (however, the R version offers an exact test as well). Here, we use the R version. The argument, `correct`, set to FALSE, will not use a continuity correction for the asymptotic test.

```
mantelhaen.test(table.6.9.array, correct=F)
```

```
        Mantel-Haenszel chi-squared test without continuity correction

data:  table.6.9.array
Mantel-Haenszel X-squared = 6.3841, df = 1, p-value = 0.01151
alternative hypothesis: true common odds ratio is not equal to 1
95 percent confidence interval:
 1.177590 3.869174
sample estimates:
common odds ratio
         2.134549
```

Thus, we reject the null hypothesis of conditional independence between Treatment and Response given Center. (The common odds ratio result is discussed later). We can also use the logit model test. This is a test that the coefficient corresponding to Treatment in a logit model is zero, given the model includes Center (see equation (6.4) in Agresti). If we use `glm`, we need to make a few modifications to the data frame. Below, I add the total to each Center/Treatment combination.

```
n<-aggregate(table.6.9$Freq, list(table.6.9$Treatment,table.6.9$Center), FUN=sum)$x
table.6.9$n<-rep(n,rep(2,16))
```

Now, we fit model (6.4). Because of the alphabetical ordering of levels of factors, the coefficient is negative. This can be changed by redoing the factor definition above.

```
options(contrasts=c("contr.treatment", "contr.poly")) # S-PLUS only
res<-glm(Freq/n~Center+Treatment, family=binomial, data=table.6.9,  weights=n, subset=
   Response=="Success")    # model 6.4
summary(res, cor=F)

Call: glm(formula = Freq/n ~ Center + Treatment, family = binomial, data = table.6.9,
   weights = n, subset = Response == "Success")

Coefficients:
                Value Std. Error    t value
(Intercept) -0.5450944  0.2929052 -1.8609927
    Centerb  2.0554344  0.4200885  4.8928604
    Centerc  1.1529027  0.4245668  2.7154802
    Centerd -1.4184537  0.6635739 -2.1375972
    Centere -0.5198903  0.5337883 -0.9739635
    Centerf -2.1469176  1.0603341 -2.0247558
    Centerg -0.7977076  0.8149166 -0.9788824
    Centerh  2.2079143  0.7195076  3.0686463
  Treatment -0.7769203  0.3066807 -2.5333195

(Dispersion Parameter for Binomial family taken to be 1 )

    Null Deviance: 93.5545 on 15 degrees of freedom

Residual Deviance: 9.746317 on 7 degrees of freedom

Number of Fisher Scoring Iterations: 5
```

The LRT is given by `anova`.

```
anova(res)

Analysis of Deviance Table

Binomial model

Response: Freq/n

Terms added sequentially (first to last)
          Df Deviance Resid. Df Resid. Dev     Pr(Chi)
     NULL                    15   93.55450
   Center  7 77.13937          8   16.41513 0.000000000
Treatment  1  6.66882          7    9.74632 0.009811426
```

And, we reject the hypothesis of conditional independence. We can also use the CMH statistic, but compare it to the exact null distribution. The p-value for this test is available in the R version of the `mantelhaen.test` function, with argument `exact`.

```
mantelhaen.test(table.6.9.array, exact=T)
```

```
        Exact conditional test of independence in 2 x 2 x k tables

data:  table.6.9.array
S = 55, p-value = 0.01336
alternative hypothesis: true common odds ratio is not equal to 1
95 percent confidence interval:
 1.137370 4.079523
sample estimates:
common odds ratio
         2.130304
```

If we assume that the log odds ratios have different directions across strata, we can test conditional independence using a logit model fit. The test is a likelihood-ratio test of model (6.5) in Agresti (only a Center effect) compared with a saturated model (Center, Treatment and Center:Treatment). With this test, we also reject the null hypothesis.

```
res2<-update(res, .~.-Treatment)            # model (6.5)
res3<-update(res, .~.+Center:Treatment)     # saturated model

anova(res2,res3,test="Chisq")

Analysis of Deviance Table

Response: Freq/n

                              Terms  Df   Res. Dev                      Test   Df   Dev.  Pr(Chi)
1                            Center   8   16.41513
2 Center+Treatment+Center:Treatment   0    0.00060 +Treatment+Center:Treatment  8 16.41  0.037
```

F. Estimation/Testing of Common Odds Ratio

If we decide that the conditional odds ratios across strata are similar enough to be combined, we can get an estimate of the common odds ratio using either the ML estimate of the Treatment coefficient in logit model (6.4, p. 231 in Agresti) or the Mantel-Haenszel estimate (p. 234 in Agresti). The latter is given by `mantelhaen.test` in R, under `common odds ratio`, along with an approximate 95% confidence interval (`95 percent confidence interval`). The ML estimate is the exponent of the Treatment coefficient from fitting logit model (6.4). Thus, the MLE of the common odds ratio is

```
# Recall p. 101:  res<-glm(Freq/n~Center+Treatment, family=binomial, data=table.6.9,
    weights=n, subset= Response=="Success")
exp(-res$coefficient[9])   # the negative relates to how factor levels are coded

 Treatment
  2.174764
```

To help us make a decision about whether to combine the odds ratios, we can perform a test of the homogeneity of odds ratios across strata. As stated in Agresti, this is a test of the goodness-of-fit of model (6.4). Thus, the test is

```
anova(res, test="Chisq")

Analysis of Deviance Table

Binomial model

Response: Freq/n

Terms added sequentially (first to last)
          Df Deviance Resid. Df Resid. Dev     Pr(Chi)
     NULL                    15   93.55450
   Center  7 77.13937         8   16.41513 0.000000000
```

```
Treatment  1  6.66882          7    9.74632 0.009811426
```

Alternatively, the Woolf test is available in the R package `vcd`, under `woolf.test`. The null hypothesis is no three-way (XYZ) interaction, or homogeneous odds ratios across Centers. The function can also be used within S-PLUS with no modification.

```
library(vcd)
woolf.test(table.6.9.array)

        Woolf-test on Homogeneity of Odds Ratios (no 3-Way assoc.)

data:  table.6.9.array
X-squared = 5.818, df = 7, p-value = 0.5612
```

G. Using Models to Improve Inferential Power

As Agresti says, ordinal test statistics in categorical data analysis usually refer to narrower, more relevant alternatives than do nominal test statistics. They also usually have more power when an approximate linear trend actually exists between the response and a predictor variable. A linear trend model (e.g., linear logit model) has fewer parameters to estimate than does a nominal model with a separate parameter for each level of the ordinal variable. The LR or Pearson goodness-of-fit statistic used to test the linear trend parameter of a linear logit model has only 1 degree of freedom associated with it, whereas the test of the corresponding nominal model against an independence model has degrees of freedom equal to one less than the number of levels of the variable. Thus, the observed value of the test statistic in the former case does not have to be as large (compared to the chi-squared distribution) as that in the second case in order to reject the independence model over the linear logit model .

Agresti uses an example on the treatment of leprosy by sulfones and streptomycin drugs to illustrate the difference in power to detect a suspected association (Table 6.11, p. 239). The degree of infiltration measures the amount of skin damage (High, Low). The response, the amount of clinical change in condition after 48 weeks, is ordinal with five categories (Worse, …, Marked Improvement). The response scores are {-1, 0, 1, 2, 3}. Agresti compares the mean change for the two infiltration levels, and notes that this analysis is identical to a trend test treating degree of infiltration as the response and clinical change as the predictor. Thus, a linear logit model has the logit of the probability of high filtration linearly related to the change in clinical condition.

In S, first we set up the data, then fit a model with a separate parameter for each change category.

```
table.6.11<-data.frame(change=factor(c("worse","stationary","slight improvement",
    "moderate improvement", "marked improvement"),
    levels=c("worse","stationary","slight improvement", "moderate improvement", "marked
    improvement")), high=c(1,13,16,15,7),  n=c(12, 66, 58, 42, 18))

res.leprosy<-glm(high/n~change, weights=n, family=binomial, data=table.6.11)
```

The test statistic for testing the null hypothesis of all change coefficients equal to zero is obtained by `anova`.

```
anova(res.leprosy, test="Chisq")

Analysis of Deviance Table

Binomial model

Response: high/n

Terms added sequentially (first to last)
       Df Deviance Resid. Df Resid. Dev   Pr(Chi)
```

```
  NULL                     4   7.277707
change  4 7.277707         0   0.000000 0.1219205
```

With p = 0.12, this test does not reject an independence model. Now, we fit a linear logit model using scores c(-1, 0, 1, 2, 3), and test it against an independence model.

```
resLL.leprosy<-glm(high/n~c(-1,0,1,2,3), weights=n, family=binomial,data=table.6.11)

anova(resLL.leprosy, test="Chisq")

Analysis of Deviance Table

Binomial model

Response: high/n

Terms added sequentially (first to last)
                 Df Deviance Resid. Df Resid. Dev     Pr(Chi)
          NULL                       4   7.277707
c(-1, 0, 1, 2, 3)  1 6.651426         3   0.626282 0.009907651
```

We do reject the independence model here, in favor of the linear logit model that hypothesizes more positive change with higher filtration.

H. Sample Size and Power Considerations

The power of a two-sample binomial test of two proportions and the sample sizes needed to attain a particular power for this test can be computed using built-in functions in S-PLUS. The same functions are not available in R, but Harrell's `Hmisc` library has similar functions. The functions do compute power for unequal sample sizes.

To compute the power of a two-sided 0.05-level test comparing two proportions, where the sample sizes are both 25, and the expected absolute difference in proportions is 0.10, we can use the functions `binomial.sample.size` (S-PLUS) or `bpower` (R and S-PLUS). We use the former first.

```
binomial.sample.size(n1=25, n2=25, p=.6, p2=.7, alpha=.05, alternative="two.sided")

   p1  p2 delta alpha      power n1 n2 prop.n2
1 0.6 0.7   0.1  0.05 0.06137109 25 25       1
```

The argument `p2` in `binomial.sample.size` is the proportion for the second group. The argument `p` is the proportion for the first group (it is not `p1`, in case there is only one sample).

With different sample sizes, 20 and 25, we expect the power to decrease.

```
binomial.sample.size(n1=20, n2=25, p=.6, p2=.7, alpha=.05,alternative="two.sided")

   p1  p2 delta alpha      power n1 n2 prop.n2
1 0.6 0.7   0.1  0.05 0.05610319 20 25    1.25
```

The formulae used by `binomial.sample.size` come from Fleiss (1981), and even with equal sample sizes, they are not exactly the same as those appearing in Agresti p. 240-242. `bpower` more closely matches Agresti (and the function contains less overhead).

With equal sample sizes, `bpower` gives

```
bpower(n=50, p1=.6, p2=.7, alpha=0.05)

     Power
 0.1135017
```

```
bpower(n=200, p1=.6, p2=.7, alpha=0.05)

     Power
 0.3158429
```

where we assume the 0.10 difference is divided as p1 = 0.6 and p2 = 0.7, and `n` is the sum of n1 and n2 (with n1 = n2). `bpower` only computes power for a two-sided alternative.

With n1 = 20 and n2 = 25, we get slightly lower power

```
bpower(n1=20,n2=25,p1=.6,p2=.7)

     Power
 0.1084922
```

`binomial.sample.size` also computes sample size needed to achieve a given power. The function `bsamsize` in package Hmisc does as well. Again, the latter is closer to Agresti's formula on p. 242, which he says is an underestimate of the sample size.

To compute sample size required to achieve 90% power with p1 = 0.6 and p2 = 0.7, and an 0.05-level test, we need close to 500 subjects for each group.

```
binomial.sample.size(p=.6, p2=.7, alpha=.05,alternative="two.sided", power=.9)

   p1  p2 delta alpha power  n1  n2 prop.n2
1 0.6 0.7   0.1  0.05   0.9 497 497       1

bsamsize(power=.9, p1=.6, p2=.7, alpha=0.05)

       n1       n2
 476.0072 476.0072
```

Sample size calculations for logistic regression can easily be programmed into a simple S function, following the formulas on p. 242 in Agresti.

Power for a chi-squared test can be computed using the following function in S-PLUS:

```
chipower.f<-function(p=NULL,pm=NULL,n,alpha,df, pearson=T, fit=NULL)
{
   # S-PLUS only
   if(pearson) nc<-n*.pearson.x2(observed=p,expected=pm)$X2
      else nc<-n*fit$deviance
   1-pchisq(qchisq(1-alpha,df),df=df,ncp=nc)
}
```

where `p` is a matrix or vector of true proportions, `pm` are the expected probabilities under the null hypothesis, and fit is a glm fit (only used for LR chi-squared). In R, we use

```
chipower.f<-function(x=NULL, pm=NULL, n=NULL, alpha, df, pearson=T, fit=NULL)
{
   # R only
    if(pearson) nc<-chisq.test(x=x,p=pm)$statistic
        else nc<-n*fit$deviance
    1-pchisq(qchisq(1-alpha,df),df=df,ncp=nc)
}
```

where `x` is a matrix or vector of true counts, and the remaining arguments have the same definitions.

As a first example, I use Table 6.13 in Agresti. This table and its "true" joint probabilities from the physician are input below. Then I fit two models, one with the standard and new therapies independently influencing the response, and one model with standard therapy alone influencing the response. We will compute the power for detecting an independent effect of the new therapy on response, over and above that from the standard therapy.

```
table.6.13<-data.frame(expand.grid(New = c("very worry", "somewhat",
   "reassuring"),Standard=c("worry", "reassure")),
prop=10000*c(.04*.4, .08*.32,.04*.27,.02*.3,.18*.22,.64*.15))

weight<-10000*c(0.04,0.08,0.04,0.02,0.18,0.64)

fit1<-glm(prop/weight~Standard, data=table.6.13, family="binomial",weights=weight)
fit2<-glm(prop/weight~Standard+New, data=table.6.13, family="binomial",
   weights=weight)
```

Now, I compute power for n = 400, 600, and 1000 with alpha = 0.05. First, I compute the difference in residual deviances between the two models using the `anova` function.

```
res<-list(deviance=anova(fit1,fit2)$Deviance[2]/10000)
chipower.f(fit=res, alpha=.05, df=2, pearson=F, n=c(400,600,1000))

[1] 0.3466997 0.4948408 0.7266697
```

As a second example of power computations, I use the data in Table 2.5 as though they were the true cell probabilities and compute power for a test of independence between smoking and lung cancer.

```
table.2.5<-data.frame(expand.grid(Smoker=c("yes","no"),
   Lung.Cancer=c("Cases","Controls")),count=c(688,21,650,59))
```

The expected probabilities under independence can be computed using formula in Section 3.2 of Agresti or found in R using the function `expected` from library `vcd`, which computes expected *counts*. Actually, the `expected` function can be sourced into S-PLUS with no modification. To use it, we need an array, however. So, we first change the data frame into an array.

```
table.2.5.array<-design.table(table.2.5)
#R: table.2.5.array<-xtabs(count~Smoker+Lung.Cancer,data=table.2.5)
```

Then, we apply `expected`, and get the power for a Pearson chi-square test assuming a total sample size of 1,418, which happens to be the same as the observed sample size.

```
fit<-expected(table.2.5.array)
chipower.f(p=table.2.5.array/1418, pm=fit/1418, n=1418, alpha=.05, df=1)  # S-PLUS
[1] 0.992105

# R: chipower.f(x=table.2.5.array, pm=fit, alpha=.05,df=1)
[1] 0.9892372
```

In this case, the noncentrality parameter is about 19. So, according to Table 6.12 in Agresti, the power should be between 0.972 and 0.998.

For other sample sizes, in S-PLUS we just change the `n` argument to `chipower.f`. In R, for the Pearson chi-squared test power we must specify the true cell counts using the true cell probabilities and our specified sample size.

I. Probit and Complementary Log-Log Models

1. Probit Models

As described in Section 6.6 in Agresti, the idea behind probit models is that there is a latent tolerance value underlying each binary response. The tolerance is subject-specific. When a linear combination of the predictor variables is high enough to exceed an individual's tolerance, the binary response becomes 1 and remains there. Otherwise, it remains at zero. The tolerance is a continuous random variable. When it has a normal distribution, we get a probit model. Probit models are used in toxicological experiments where the predictor is dosage. So, when the dosage exceeds a threshold (tolerance), the response is death.
In GLIM terminology, the probit link function is the inverse CDF of the standard normal distribution (equation 6.11 in Agresti). Agresti gives some differences between probit link and logit link for binomial regression. Fitting the probit model is fitting a GLIM, so no new estimation procedures are introduced. Estimates from Newton-Raphson and Fisher scoring will have slightly different estimated standard errors because observed and expected information differ in the probit model. The probit is not the canonical link for a binomial model.

Agresti fits a probit model to the beetle mortality data in Table 6.14.

```
table.6.14<-data.frame(log.dose=c(1.691,1.724,1.755,1.784,1.811,1.837,1.861,1.884),
n=c(59,60,62,56,63,59,62,60),
y=c(6,13,18,28,52,53,61,60))
```

Here, we use Fisher scoring to fit the probit model. If we wanted to use Newton-Raphson, we could use the function `probitreg`, defined earlier. The two methods give different standard errors.

```
options(contrasts=c("contr.treatment", "contr.poly")) # S-PLUS only
(res.probit<-glm(y/n~log.dose,weights=n,family=binomial(link=probit),
   data=table.6.14))

Coefficients:
 (Intercept) log.dose
   -34.95563 19.74073

Degrees of Freedom: 8 Total; 6 Residual
Residual Deviance: 9.986957
```

2. Complementary Log-Log Models

The logit and probit links are symmetric about 0.5, meaning that the probability of success approaches 0 at the same rate as it approaches 1. The complementary log-log link is asymmetric. Its probability of success approaches 0 slowly but approaches 1 sharply.

For the beetle mortality data, we can see from the observed proportions that they quickly increase to 1 after a dosage of 1.784. Thus, the complementary log-log should be a better fit than a probit.

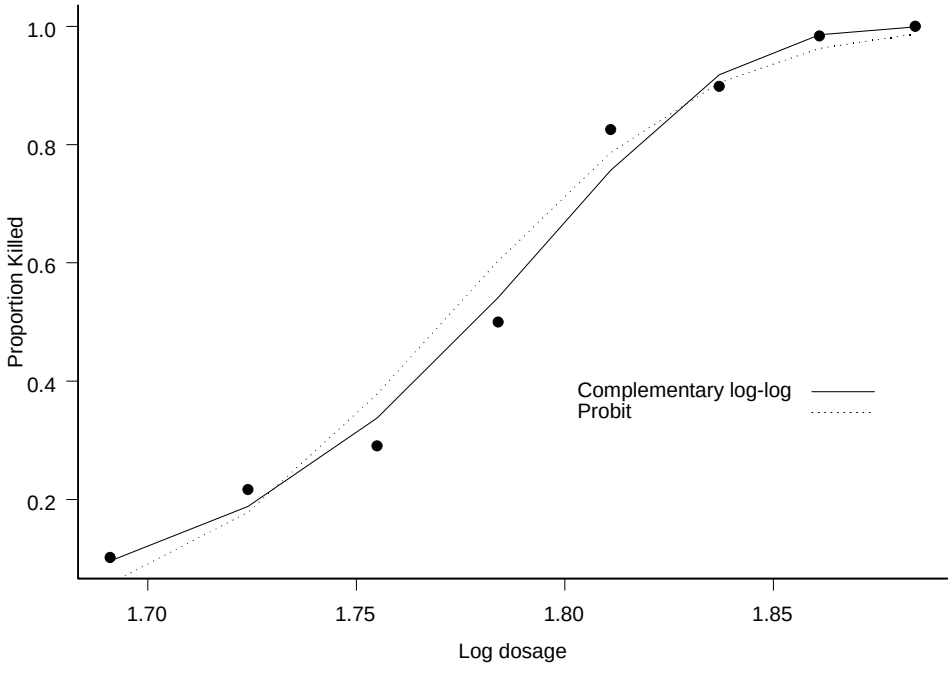
```
(res.cloglog <- glm(y/n ~ log.dose, weights = n, family = binomial(link = cloglog),
   data = table.6.14))

Coefficients:
 (Intercept) log.dose
   -39.52232 22.01478

Degrees of Freedom: 8 Total; 6 Residual
Residual Deviance: 3.514334
```

The LR statistic is much lower for complementary log-log link. Here are the fitted values and Figure 6.6.

```
data.frame(table.6.14, fitted.probit = round(n * fitted(res.probit), 1),
fitted.cloglog = round(n * fitted(res.cloglog),1))
```

```
  log.dose  n  y fitted.probit fitted.cloglog
1    1.691 59  6           3.4            5.7
2    1.724 60 13          10.7           11.3
3    1.755 62 18          23.4           20.9
4    1.784 56 28          33.8           30.3
5    1.811 63 52          49.6           47.7
6    1.837 59 53          53.4           54.2
7    1.861 62 61          59.7           61.1
8    1.884 60 60          59.2           59.9
```

```
plot(table.6.14$log.dose,table.6.14$y/table.6.14$n, pch=16, xlab="Log dosage",
   ylab="Proportion Killed", bty="L", axes=F)
axis(1, at=seq(1.7, 1.9, .05))
axis(2, at=seq(0,1,.2))
lines(table.6.14$log.dose, fitted(res.probit), lty=2)
lines(table.6.14$log.dose, fitted(res.cloglog), lty=1)
key(x=1.8, y=.4, text=list(c("Complementary log-log", "Probit")),
   lines=list(type=c("l", "l"), lty=c(1,2)), border=F,text.width.multiplier=.8)
# R: legend(x=1.8,y=.4,legend=c("Complementary log-log", "Probit"),lty=c(1,2),
   cex=.85, text.width=1)
```



J. Conditional Logistic Regression and Exact Distributions

1. Small-sample Conditonal Inference for 2x2 Contingency tables

Small-sample conditional tests for 2 x 2 tables use Fisher's Exact test. See the discussion in Chapter 3.

2. Small-sample Conditonal Inference for Linear Logit Models

Small-sample conditional inference for the linear logit model is an exact conditional trend test, and the conditional distribution of the cell counts under the null hypothesis of $\beta = 0$ is multiple hypergeometric. Agresti applies an exact test to Table 5.3 on maternal alcohol consumption and infant malformation. This

data set has a lot of cases, but the table is sparse in that there are few successes or cases. Thus, ordinary maxmimum likelihood inference (with its asymptotic basis) breaks down.

To do conditional logistic regression in S, one can use the `coxph` function (or the `clogit` function in R, which is just a wrapper for `coxph`). However, with more than about 500 cases (in my experience), the function hangs "forever". Thus, we turn to approximations to conditional logistic regression for large samples. There are S functions available for doing approximate conditional logistic regression on a scalar parameter of interest. One is `cond.glm` (normal approximation) from Alessandra Brazzale (see http://www.ladseb.pd.cnr.it/~brazzale/lib.html#ins). Another is `cox.test` (for a normal approximation) and `cpvalue.saddle` and `cl.saddle` (for a saddle-point approximation) from Chris Lloyd (Lloyd, 1999). These latter functions are apparently available on the Wiley website (http://www.wiley.com), but I have not been able to locate them yet. I contacted Professor Lloyd, via the Wiley website, for copies.

First, I show how to use `coxph` and `clogit` for an exact conditional test. However, the problem is much too large to use this method. So, I do not give results. The response value must be binary ("cases" indicate successes). Here I create a data frame with `case` as the response, and `alcohol` as a numeric explanatory variable.

```
# recall from chapter 5
malformed<-c(48,38,5,1,1)
n<-c(17066,14464,788,126,37)+malformed

temp<-length(10)
temp[rep(c(T,F),5)]<-malformed
temp[rep(c(F,T),5)]<-n-malformed
table.5.3<-data.frame(Alcohol=rep(c(0,.5,1.5,4,7), n), case=rep(rep(c(1,0),5),temp))
```

Now, we use `coxph`. The "`time`" argument for the function `Surv` is just a vector of ones with as many values as persons. `case` is the `status` (whether success or failure) for the `Surv` object.

```
fit<-coxph(Surv(rep(1,sum(temp)),case)~Alcohol, table.5.3, method="exact")
#R: library(survival)
#R: fit<-clogit(case~Alcohol, data=table.5.3, method="exact")
```

However, the computation time is excessive. `clogit` offers a test that uses an approximate conditional likelihood. Here the conditioning is on the sufficient statistic for the intercept, in order to remove that parameter, and therefore get an exact test for the alcohol coefficient.

```
# R only
fit<-clogit(case~Alcohol, data=table.5.3, method="approximate")

Call:
clogit(case ~ Alcohol, data = table.5.3, method = "approximate")


         coef exp(coef) se(coef)    z     p
Alcohol 0.315      1.37    0.124 2.54 0.011

Likelihood ratio test=4.23  on 1 df, p=0.0397  n= 32574
```

Alessandra Brazzale's original S-PLUS function `cond.glm` was written for S-PLUS 4.5 and 2000 (as well as for unix), but it can be modified for sourcing into S-PLUS 6.0. On my website, I have one method for doing so. However, it is very quick and dirty. So, you probably should make changes if you seriously want to use this function in S-PLUS 6.x. In addition, she has recently provided an R package called `cond`, that contains the function `cond.glm`.

The `cond.glm` function takes a `glm.object` and an `offset` that gives the name of the object that is the explanatory variable of interest (for which you want the inference). The function returns the approximate

conditional estimate of the coefficient and its standard error. A `summary` function gives hypothesis tests about this coefficient.

```
# R: library(cond)
(fit<-cond(glm(formula=case~Alcohol, family=binomial, data=table.5.3),
   offset=Alcohol))

summary(fit, test=T)    # R output

Formula:  case ~ Alcohol
 Family:  binomial
 Offset:  Alcohol

          Estimate   Std. Error
uncond.      0.3166       0.1254
cond.        0.3165       0.1253

Confidence intervals
--------------------
 level = 95 %
                                          lower two-sided  upper
MLE normal approx.                      0.07069           0.5624
Cond. MLE normal approx.                0.07095           0.5621
Directed deviance                       0.01870           0.5236
Modified directed deviance              0.03626           0.5312
Modif. direct. deviance (cont. corr.)   0.02301           0.5357

Diagnostics:
-----------
     INF       NP
0.097872 0.003178

 Approximation based on 20 points
```

The p-values for the normal approximations are distinctly smaller than the exact p-values in Agresti.

Lloyd's (1999) function `cl.saddle` computes a saddlepoint approximation to the conditional log-likelihood for any parameter in a GLIM, given a range of values for the parameter. The function `cpvalue.saddle` computes an approximation to the conditional p-value in testing that the parameter is zero. The approximations are based on a saddlepoint approximation to the conditional density of the sufficient statistic for the parameter of interest.

To use the functions in S-PLUS 6.x, we must make the following two modifications. The first modification changes the function for computing the log of the determinant. The second changes the way a very large diagonal matrix is handled in a matrix multiplication. So, in `cpvalue.saddle`, change

```
   ldv.null <-  - log.determinant(t(X.null) %*% diag(null$weights) %*% X.null)
```

to

```
   ldv.null <-  - log(det((t(X.null) * null$weights) %*% X.null))
```

and change

```
   ldv <-  - sum(log(eigen(t(X) %*% diag(fit$weights) %*% X)$value))
```

to

```
   ldv <-  - sum(log(eigen((t(X) * fit$weights) %*% X)$value))
```

In `cl.saddle`, change

```
       ldv.null[i] <- - sum(log(eigen(t(X.null)%*%diag(null$weights)%*% X.null)$values))
```

to

```
       ldv.null[i] <- - sum(log(eigen((t(X.null)*null$weights)%*%X.null)$values))
```

Modifications within R would be similar.

To use `cl.saddle` on `table.5.3`, which is quite large, we type the following. I choose to compute `B=20` points of the conditional log-likelihood, within the range –1 to 1 of the parameter value. The `term` argument specifies the parameter of interest.

```
attach(table.5.3)
(approx.points<-cl.saddle(y=case, formula=case~Alcohol, term="Alcohol", family=
   binomial, u=1, l=-1, B=20))
```

```
$x:
 [1] 0.009 0.114 0.220 0.325 0.430 0.535 0.641 0.746 0.851 0.956 1.062 1.167 1.272 1.377 1.483
    1.588 1.693 1.798 1.904 2.009

$y:
 [1]   -2.0245273   -0.9880964   -0.2555597    0.0000000   -0.4881732   -2.1817449   -5.8389314
    -12.3404829  -22.6823900
[10]  -37.5292684  -57.2109348  -80.9817436 -108.3950206 -138.8544144 -172.2877420 -207.6226743 -
    244.9780080 -284.1413107
[19] -325.2938305 -367.4350287
```

Although the x values are not in the range we specified, we can see that the maximum of the conditional log-likelihood is attained at around 0.325 or so, which coincides with previous analyses. We can plot these, if we like.

Now, to get the p-value, we use `cpvalue.saddle`, and take the complement of its approximation to the CDF.

```
1-cpvalue.saddle(y=case, formula=case~Alcohol, term="Alcohol",family=binomial, H0=0)
detach(table.5.3)
```

```
    Alcohol
 0.02273689
```

3. Small-sample Conditonal Inference for 2 x 2 x K Tables

For 2 x 2 x K tables, where the separate 2 x 2 tables belong to K different strata, an exact test of the coefficient on the risk factor conditions on the row and column totals within each table. Then, within each table, the null distribution of the test statistic (the count in the first cell) is hypergeometric. In this case, the null hypothesis is conditional independence of X and Y given Z (strata). The product of the hypergeometric mass functions from the separate strata gives the joint null distribution of the cell counts. Agresti gives more details on p. 254-255 (see also Chapter 3 in Agresti).

Agresti uses a data set on promotion decisions for similarly tenured government computer specialists for three different months (strata). Conditional independence of promotion decision and race is equivalent to a test of the race coefficient in a logistic regression model that includes month and race. However, ML estimation is not recommended due to the prescense of three zero-count cells. Exact conditional tests of independence for these tables can be carried out using `mantelhaen.test` in R, with argument `exact=T`. In S-PLUS, the exact test is not an option.

```
table.6.15<-
data.frame(expand.grid(race=c("black","white"),promote=c("yes","no"),month=c("july","a
ugust","september")), count=c(0,4,7,16,0,4,7,13,0,2,8,13))
```

Recall that we need to have a three-dimensional array for `mantelhaen.test`.

```
# R only (one-sided test; use alternative="two.sided" for two-sided test)
```

```
mantelhaen.test(xtabs(count~month+race, data=table.6.15), exact=T, alternative=
"less")

        Exact conditional test of independence in 2 x 2 x k tables

data:  xtabs(count ~ race + promote + month, data = table.6.15)
S = 0, p-value = 0.02566
alternative hypothesis: true common odds ratio is less than 1
95 percent confidence interval:
 0.0000000 0.7795136
sample estimates:
common odds ratio
                0
```

The result using S-PLUS and the `correct=F` argument is

```
mantelhaen.test(design.table(table.6.15), correct=F)

    Mantel-Haenszel chi-square test without continuity correction

data:  design.table(table.6.15)
Mantel-Haenszel chi-square = 4.5906, df = 1, p-value = 0.0321
```

A 95% two-sided confidence interval on the odds ratio is printed automatically.

```
# R only
mantelhaen.test(xtabs(count~race+promote+month, data=table.6.15), exact=T, alternative
= "two.sided")

        Exact conditional test of independence in 2 x 2 x k tables

data:  xtabs(count ~ race + promote + month, data = table.6.15)
S = 0, p-value = 0.05625
alternative hypothesis: true common odds ratio is not equal to 1
95 percent confidence interval:
 0.000000 1.009031
sample estimates:
common odds ratio
                0
```

A final example is given where the table displays “separation”. That is, the cases all correspond to one level of the risk factor. In these situations, as Agresti says, maximum likelihood estimation gives infinite estimates. Thus, exact inference is needed. Since the number of observations is quite large, we can’t use `coxph`. Even if we take advantage of the fact that we really have a 2x2 table in Cephalexin and Diarrhea (as mentioned by Agresti, bottom of p. 256), we still have over 1,000 observations. However, with the 2x2 table, we can use Fisher’s Exact Test. Note that S-PLUS won’t handle this data set because it is too large (> 200 counts total). However, R handles it.

```
table.6.16<-expand.grid(Ceph=factor(c(0,1)), Age=factor(c(0,1)),
      Stay= factor(c(0,1)), case=factor(c(0,1)))
table.6.16$count<-c(385,0,789-3,0,233-5,0,1081-47,0,0,0,3,0,5,0,47,5)

# R only
fisher.test(xtabs(count~Ceph+case,data=table.6.16,subset=(Age==1 & Stay==1)))

        Fisher's Exact Test for Count Data

data:
p-value = 2.084e-07
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 19.30481      Inf
sample estimates:
```

```
odds ratio
       Inf
```

Note that we get the confidence interval as well.

K. Bias-reduced Logistic Regression

In cases of quasi or complete separation, a different kind of estimation can be done instead of ordinary maximization. Bias-reduced logistic regression (Firth, 1993, as cited in R help file for package `brlr`) maximizes a penalized likelihood with penalty function, Jeffreys invariant prior. This leads to less biased estimates that are always finite. According to Firth, the bias reduction (toward zero) can be quite noticeable for problems that display separation. Here, we fit the main-effects only logistic regression model from subsection J to the diarrhea data, using `glm` (ML) and `brlr` (penalized ML). Note the difference in the coefficient estimate for Cephalexin, as well as its smaller standard error from the penalized estimation. The large magnitude of the Ceph estimate and its corresponding large standard error from the `glm` fit indicate an infinite maximizer.

```
library(brlr)
fit<-glm(case~Ceph+Age+Stay, family=binomial,data=table.6.16)
fit2<-brlr(case~Ceph+Age+Stay, data=table.6.16)

summary(fit)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -6.6100     0.6928  -9.541  < 2e-16 ***
Ceph         11.6454    19.6574   0.592   0.5536
Age           0.8519     0.4741   1.797   0.0723 .
Stay          2.6785     0.5898   4.541 5.59e-06 ***
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 565.77  on 2492  degrees of freedom
Residual deviance: 475.60  on 2489  degrees of freedom
AIC: 483.6

Number of Fisher Scoring iterations: 7

summary(fit2)

Coefficients:
            Value   Std. Error t value
(Intercept) -6.3793  0.6480    -9.8444
Ceph         5.2749  1.5041     3.5071
Age          0.7716  0.4527     1.7045
Stay         2.5403  0.5523     4.5994

Deviance: 476.7722
Penalized deviance: 470.8072
Residual df: 2489
```

Chapter 7 – Logit Models for Multinomial Responses

A. Summary of Chapter 7, Agresti

In Chapter 7, Agresti discusses logit models for multinomial responses, both nominal responses and ordered responses. Multinomial logit models for nomial responses use a separate binary logit model for each pair of response categories (some of which are redundant), and are fit using maximum likelihood estimation subject to the individual response probabilities simultaneously satisfying their separate logit models.

There are several types of models for ordinal response categories. Section 7.2 discusses cumulative logit models, which use the logits of the cumulative probabilities $P(Y \le j \,|\, \mathbf{x}) = \pi_1(\mathbf{x}) + ... + \pi_j(\mathbf{x})$, $j = 1,\ldots,J$. So, the cumulative logit is the log of the odds of responding in category up to j. A cumulative logit model models all $J - 1$ cumulative logits simulataneously. A popular cumulative logit model is a proportional odds model. This model assumes the $J - 1$ logits have different intercepts, increasing in j, but otherwise the coefficients are identical. Thus, the $J - 1$ response curves have the same shape, but are shifted horizontally from one another. The term *proportional odds* applies because the log of the odds ratio of cumulative probabilities (comparing the conditions in one covariate vector to another) is proportional to the distance between the covariate vector. The same proportionality constant applies to each logit. Proportional odds models are fit using maximum likelihood estimation.

A latent variable interpretation of the proportional odds model assumes an underlying (unobserved) continuous response variable that has a logistic distribution. When this response falls between the $(j - 1)$th and jth cutpoints on the response scale, the observed discrete response value is j. Effects of explanatory variables are invariant to the choice of cutpoints for the response so that the coefficients are identical across logits.

Section 7.3 discusses cumulative link models for ordinal response categories. Here, we use a link function that relates the cumulative probabilities to a linear predictor involving the covariates. The use of the logit link is the cumulative logit model. The probit link gives the cumulative probit model. These models assume that the ordinal response categories have an underlying continuous distribution that is related to the particular link function. They also assume that the distributions of each covariate setting are stochastically ordered, so that the cumulative probabilities given one covariate setting are always at least as great or at least as less as the cumulative probabilities of another covariate setting. If this is not so, then one covariate setting may differ in dispersion of responses than another covariate setting.

Other models for ordinal responses are discussed in Section 7.4. These include adjacent-categories logit models, continuation-ratio logit models, and mean response models. The adjacent-categories logits are the $J - 1$ logits formed by the log odds of the probability of response falling in category j given that the response falls within either category j or $j + 1$, $j = 1,\ldots,J - 1$. Adjacent-categories logit models differ from cumulative logit models in that effects of explanatory variables refer to individual response categories and not cumulative groups. They also do not assume an underlying latent continuous response.

Continuation-ratio logits, as given in Agresti, are the logits of the conditional probabilities that the response falls in the jth category given that it falls at least in the jth category for $j = 1,\ldots,J - 1$. The full likelihood is the product of multinomial mass functions, which can each be represented as a product of binomial mass functions, leading to an easy way to estimate these models.

Finally, mean response models for ordered responses are linear regression models for ordinal responses, represented by numerical scores. The linear probability model is a special case. They can be fit using maximum likelihood estimation, where the sampling model is product multinomial.

To test conditional independence in I x J x K tables, where one factor Z is conditioned, one can use multinomial models such as cumulative logit models or baseline-category logit models depending on whether the response Y is ordinal or nominal, respectively. The test of homogenous XY association across levels of Z is then a test of the inclusion of a term or terms involving X.

Generalized Cochran-Mantel Haenszel (CMH) tests can be used to test conditional independence for I x J x K tables with possibly ordered categories. With both X and Y ordinal, the

generalization is Mantel's M^2 statistic. Some versions of the generalized CMH statistic correspond to score tests for various multinomial logit models.

Discrete-choice multinomial logit models use a different set of values for a categorical explanatory variable for different response choices. Conditional on the choice set (e.g., choosing which type of transportation one prefers), the probability of selecting option *j* is a logistic function of the explanatory variables. The model can also incorporate explanatory variables that are characteristic of the person doing the choosing. Fitting these models using maximum likelihood can be done using the `coxph` function in R or S-PLUS, as the likelihood has the same form as the Cox partial likelihood. This can be seen from equation (7.22) in Agresti.

B. Nominal Responses: Baseline-Category Logit Models

When responses are nominal, for a given covariate pattern, $\mathbf{x}$, the vector of counts at the J response categories is assumed multinomial with probability vector $\{\pi_1(\mathbf{x}),...,\pi_J(\mathbf{x})\}$. The $J-1$ unique logits pair each of $J-1$ response categories with a baseline category (say, the Jth category). As shown on p. 268 of Agresti, from these $J-1$ logits that compare each category with a baseline, we can estimate logits comparing any two categories.

Agresti uses the Alligator Food Choice data set to fit a multinomial logit model. The measured response is the category of primary food choice of 219 alligators caught in four Florida lakes (fish, invertebrate, reptile, bird, other). The explanatory variables are all categorical: L = lake of capture, G = gender, S = size ($\leq$ 2.3 m, > 2.3 m). The data can be entered into S using the following commands. The `levels` argument codes the first level indicated as the baseline category.

```
food.labs<-factor(c("fish","invert","rep","bird","other"),levels=c("fish","invert",
   "rep", "bird","other"))
size.labs<-factor(c("<2.3",">2.3"),levels=c(">2.3","<2.3"))
gender.labs<-factor(c("m","f"),levels=c("m","f"))
lake.labs<-factor(c("hancock","oklawaha","trafford","george"),levels=c("george",
   "hancock", "oklawaha","trafford"))

table.7.1<-expand.grid(food=food.labs,size=size.labs,gender=gender.labs,
   lake=lake.labs)
temp<-
   c(7,1,0,0,5,4,0,0,1,2,16,3,2,2,3,3,0,1,2,3,2,2,0,0,1,13,7,6,0,0,3,9,1,0,2,0,1,0,1,0
   ,3,7,1,0,1,8,6,6,3,5,2,4,1,1,4,0,1,0,0,0,13,10,0,2,2,9,0,0,1,2,3,9,1,0,1,8,1,0,0,1)

table.7.1<-structure(.Data=table.7.1[rep(1:nrow(table.7.1),temp),], row.names=1:219) #
   structure gives back the row names
```

We fit the models from Table 7.2 using, first, the `multinom` function from library `nnet` in both S-PLUS and R. This function fits multinomial logit models with nominal response categories.

```
library(nnet)
options(contrasts=c("contr.treatment","contr.poly"))
fitS<-multinom(food~lake*size*gender,data=table.7.1)     # saturated model
fit0<-multinom(food~1,data=table.7.1)                    # null
fit1<-multinom(food~gender,data=table.7.1)               # G
fit2<-multinom(food~size,data=table.7.1)                 # S
fit3<-multinom(food~lake,data=table.7.1)                 # L
fit4<-multinom(food~size+lake,data=table.7.1)            # L + S
fit5<-multinom(food~size+lake+gender,data=table.7.1)     # L + S + G
```

The following give the LR chi-squared statistics for each model.

```
deviance(fit1)-deviance(fitS)     # or fit1@deviance - fitS@deviance (S-PLUS only)
deviance(fit2)-deviance(fitS)
deviance(fit3)-deviance(fitS)
```

```
deviance(fit4)-deviance(fitS)
deviance(fit5)-deviance(fitS)
deviance(fit0)-deviance(fitS)
```

Collapsing over gender gives

```
# options(contrasts=c("contr.treatment","contr.poly"))
fitS<-multinom(food~lake*size,data=table.7.1)      # saturated model
fit0<-multinom(food~1,data=table.7.1)              # null
fit1<-multinom(food~size,data=table.7.1)           # S
fit2<-multinom(food~lake,data=table.7.1)           # L
fit3<-multinom(food~size+lake,data=table.7.1)      # L + S

deviance(fit1)-deviance(fitS)     # or fit1@deviance - fitS@deviance (S-PLUS only)
deviance(fit2)-deviance(fitS)
deviance(fit3)-deviance(fitS)
deviance(fit0)-deviance(fitS)

 [1] 66.2129
 [1] 38.16723
 [1] 17.07983
 [1] 81.36247
```

The fitted values in Table 7.3 from the L+S model can be found using the fitted response probabilities, which are given in a slot (attribute in R) from the appropriate object of class `nnet` (here, `fit3`). The correct counts to multiply by the probabilities are the row counts from Table 7.3. Thus, first, we get these row counts using `tapply`. (I use `factor` on the necessary variables from `table.7.1` in order to reorder the levels so that they are in the same order as Table 7.3.)

```
marg.counts <- tapply(table.7.1$food, list(factor(table.7.1$size, levels = c("<2.3",
    ">2.3")),factor(table.7.1$lake,
    levels =c("hancock", "oklawaha", "trafford", "george"))), length)
```

The row names of the fitted table of counts will come from the dimension names of `marg.counts`. We start by using `expand.grid` on the `dimnames` of `marg.counts`. However, we will soon concatenate across rows to get row names.

```
row.names.71 <- rev(expand.grid(dimnames(marg.counts)))
```

Here are the fitted counts, rounded to 1 decimal place. In S-PLUS, we extract the `fitted` slot (which is a matrix) from the `fit3` object of class `nnet`, using `@` along with the name of the slot (In R, we can use `fitted()`. Actually, in S-PLUS you can as well.). The resulting matrix has duplicated rows, which we eliminate using the `duplicated` method for data frames. Note that I had to coerce `fit3@fitted` to a data frame in order to do this.

```
fitted.counts <- round(as.vector(marg.counts)*
    fit3@fitted[!duplicated(as.data.frame(fit3@fitted)),  ], 1) # S-PLUS only

# Both R and S-PLUS:
fitted.counts<-round(as.vector(marg.counts)* fitted(fit3)[!duplicated(as.data.frame(
    fitted(fit3))),],1)
```

Now, we can print the table of fitted counts. I used the `apply` function to concatenate rows of `row.names.71`, using `paste`. Also, to have the row names put to correct use, we must have a data frame as our `.Data` argument.

```
structure(.Data = as.data.frame(fitted.counts), row.names = apply(row.names.71, 1,
    paste, collapse = " "))

              fish invert rep bird other
```

```
 hancock <2.3 20.9    3.6 1.9  2.7   9.9
 hancock >2.3  9.1    0.4 1.1  2.3   3.1
oklawaha <2.3  5.2   12.0 1.5  0.2   1.1
oklawaha >2.3 12.8    7.0 5.5  0.8   1.9
trafford <2.3  4.4   12.4 2.1  0.9   4.2
trafford >2.3  8.6    5.6 5.9  3.1   5.8
  george <2.3 18.5   16.9 0.5  1.2   3.8
  george >2.3 14.5    3.1 0.5  1.8   2.2
```

Each of the 4 binary logit models has a set of estimated effects or coefficients. These can be extracted using the `summary` function, as here. Based on the way the factor levels were given in the definition of `table.7.1`, the category fish is the baseline category, and each logit model compares the odds of selecting another food to the odds of selecting fish.

```
library(MASS)        # needed for vcov function
summary(fit3, cor = F)

Coefficients:
       (Intercept)       size lakehancock lakeoklawaha laketrafford
invert   -1.549021  1.4581457  -1.6581178  0.937237973     1.122002
   rep   -3.314512 -0.3512702   1.2428408  2.458913302     2.935262
  bird   -2.093358 -0.6306329   0.6954256 -0.652622721     1.088098
 other   -1.904343  0.3315514   0.8263115  0.005792737     1.516461

Std. Errors:
       (Intercept)      size lakehancock lakeoklawaha laketrafford
invert   0.4249185 0.3959418   0.6128465    0.4719035    0.4905122
   rep   1.0530583 0.5800207   1.1854035    1.1181005    1.1163849
  bird   0.6622971 0.6424863   0.7813123    1.2020025    0.8417085
 other   0.5258313 0.4482504   0.5575446    0.7765655    0.6214372
```

To estimate response probabilities using values on the explanatory variables that (together) did not appear as a data record, you can use `predict`, with `newdata`, a data frame containing the labeled explanatory values. In S-PLUS (but not R), when we have categorical variables entered as factors in the original fit, we have to specify the prediction values as being levels of those factors. This can be done by using `factor` along with the original levels. So, to estimate the probability that a large alligator in Lake Hancock has invertebrates as the primary food choice, we use in S-PLUS

```
# S-PLUS: options(contrasts=c("contr.treatment", "contr.poly"))
predict(fit3, type="probs", newdata=data.frame(size=factor(">2.3", levels=c(">2.3",
       "<2.3")), lake=factor("hancock",levels=c("george", "hancock", "oklawaha",
       "trafford"))))      # S-PLUS or R

      fish     invert        rep      bird     other
 0.5701841 0.02307664 0.07182898 0.1408967 0.1940136
```

and in R, we can use

```
# R only
predict(fit3, type="probs", newdata=data.frame(size=">2.3", lake="hancock"))

      fish     invert        rep       bird      other
0.57018414 0.02307664 0.07182898 0.14089666 0.19401358
```

Obviously, the specification in R is how we would want to do it!

To get the estimated response probabilities for all combinations of levels of the predictors, use a call to `expand.grid` as the `newdata`. For example,

```
predictions<-predict(fit3, type = "probs", newdata = expand.grid(size = size.labs,
lake = lake.labs))
```

```
cbind(expand.grid(size = size.labs, lake = lake.labs), predictions)

  size     lake      fish     invert        rep        bird      other
1 <2.3  hancock 0.5352844 0.09311222 0.04745855 0.070402771 0.25374210
2 >2.3  hancock 0.5701841 0.02307664 0.07182898 0.140896663 0.19401358
3 <2.3 oklawaha 0.2581899 0.60188001 0.07723295 0.008820525 0.05387662
4 >2.3 oklawaha 0.4584248 0.24864188 0.19484366 0.029424140 0.06866547
5 <2.3 trafford 0.1843017 0.51682299 0.08877041 0.035897985 0.17420697
6 >2.3 trafford 0.2957470 0.19296047 0.20240167 0.108228505 0.20066230
7 <2.3   george 0.4521217 0.41284674 0.01156715 0.029664777 0.09379957
8 >2.3   george 0.6574619 0.13968168 0.02389991 0.081046954 0.09790956
```

The same multinomial model can be fit using `lcr` in R package ordinal from P. Lindsey. However, that function is much more complicated to use than `multinom` because it fits more flexible models. We will see it in the next sections for ordinal responses.

Another function that is as easy to use as `multinom` is `vglm` from R package `VGAM` (and soon to be package for S-PLUS 6.X, although it exists for the Unix versions). The commands for fitting the main effects model above is

```
library(vgam)
fit.vglm<-vglm(food~size+lake, multinomial, data=table.7.1)
-coef(fit.vglm, matrix=T)  # need to negate coefficients to match those of Agresti
```

C. Ordinal Responses: Cumulative Logit Models

This section covers fitting models to data with ordinal responses using cumulative logits. Cumulative logits are logits of cumulative probabilities. Since the responses are ordinal, a cumulative probability that the response is less than a certain category has meaning. Cumulative logit models simultaneously fit all $J - 1$ logit models for the J categories of the response. The individual category probabilities are found by subtracting adjacent cumulative probabilities.

A proportional odds model assumes the same covariate effects for each logit, but different intercepts. Thus, it has fewer parameters than an analogous multinomial logit model. There are many S-PLUS and R functions to fit these models. We will use the Mental Impairment example to illustrate each one. This data set has ordinal response categories (well, mild symptoms, moderate symptoms, impaired) to indicate the state of each human adult subject. Explanatory variables are a score on the Life Events Index and SES (1 = high, 0 = low). As the data set is available on Agresti's web site, we can just copy it to a text file and read it in.

```
table.7.5<-read.table("mental.ssc",col.names=c("mental", "ses", "life"))
```

We will illustrate using the functions `polr` (library MASS), `lrm` (library Desing), `lcr` (library ordinal), and `nordr` (library gnlm). The first two are in both R and S-PLUS. The last three are in R packages.

Function polr

As `polr` requires an ordered factor as a response, it is convenient for us to do the ordering within the data set (but, we could have done it within the model formula). So,

```
table.7.5$mental<-ordered(table.7.5$mentalC, levels=1:4, labels=c("well", "mild",
   "moderate","impaired"))
```

Also, as `polr` uses the parameterization mentioned on pp. 278-279 of Agresti (with the negative of the effect vector), we negate the explanatory variables.

```
table.7.5$ses<- -table.7.5$ses
table.7.5$life<- -table.7.5$life
```

Now, we fit the main effects model

```
library(MASS)
fit.polr <- polr(mental ~ ses + life, data = table.7.5)
summary(fit.polr)

Coefficients:
          Value Std. Error   t value
 ses  1.1112339  0.6108799  1.819071
life -0.3188611  0.1210038 -2.635134

Intercepts:
                    Value Std. Error t value
        well|mild -0.2819  0.6423    -0.4389
    mild|moderate  1.2128  0.6607     1.8357
moderate|impaired  2.2094  0.7210     3.0644

Residual Deviance: 99.0979
AIC: 109.0979
```

Function lrm

The function `lrm` in library `Design` uses the parameterization in equation (7.6) of Agresti. Thus, prior to using it, we re-negate the columns `ses` and `life`, returning them to their original values.

```
table.7.5$ses <-  - table.7.5$ses
table.7.5$life <-  - table.7.5$life
```

We also must reorder the levels of the `mental` variable, having "well" be the highest rating.

```
table.7.5$mental.rev<-ordered(table.7.5$mentalC, levels=4:1, labels=c("impaired",
   "moderate","mild","well"))
...
impaired < moderate < mild < well
```

The fit is as easy as using `polr`.

```
 (fit.lrm <- lrm(mental.rev ~ ses + life, data = table.7.5))

Logistic Regression Model

Frequencies of Responses
 impaired moderate mild well
        9        7   12   12

 Obs Max Deriv Model L.R. d.f.       P     C   Dxy Gamma Tau-a    R2 Brier
  40    8e-010       9.94    2 0.0069 0.705 0.409 0.425  0.31 0.236 0.146

              Coef   S.E. Wald Z      P
y>=moderate  2.2094 0.7210  3.06  0.0022
    y>=mild  1.2128 0.6607  1.84  0.0664
    y>=well -0.2819 0.6423 -0.44  0.6607
        ses  1.1112 0.6109  1.82  0.0689
       life -0.3189 0.1210 -2.64  0.0084
```

Function lcr

The function `lcr` in library `ordinal` for R is nowhere near as transparent to use as the above two functions. However, it will fit proportional odds models and is flexible enough to allow repeated

measurements and time-varying covariates. To use the function, we have to set up the response and covariate objects, then put them within a `repeated` object. These additional functions come from the `rmutil` library, which is loaded with `ordinal`.

```
library(ordinal)
```

First, we must have a numeric response, as well as numeric covariates. So, we use the `mentalC` variable of `table.7.5`. It also must begin at 0, not 1. (Factor covariates must be transformed to dummy coding or another numeric coding scheme. One can do this using the function `wr`, in the `rmutil` library.)

```
# set up response vector
y <- restovec(table.7.5$mentalC-1,weights=rep(1,40),type="ordinal")

# create covariat object
tcc <- tcctomat(table.7.5$ses,name="ses")
tcc <- tcctomat(table.7.5$life,name="life",oldccov=tcc)

# create a repeated object, but with no repeats here
w <- rmna(response=y,ccov=tcc)
```

Once the `repeated` object is created, the fit is easy to specify. The distribution "`prop`" means fit a proportional odds model.

```
fit.lcr<-lcr(w,distribution="prop",mu=~ses+life)

Call:
lcr(w, distribution = "prop", mu = ~ses + life)

Individual data.
Total number of individuals: 40
Number of observations:      40

Proportional odds distribution.
Transformation: identity.
Link:           logit.

Regression coefficients
             estimate    s.e.
(Intercept0)   -0.2819  0.6423
(Intercept1)    1.2128  0.6607
(Intercept2)    2.2094  0.7210
ses             1.1112  0.6109
life           -0.3189  0.1210
```

Function nordr

The function `nordr` in library `gnlm` also fits proportional odds models. It uses the function `nlm` for the maximum likelihood estimation. And, you must supply starting values. Also, it can take a `repeated` object as input for the response and environment. We use the same one created above.

```
library(gnlm)
attach(table.7.5)
```

Starting values are specified in the `pmu` and `pintercept` arguments. The `mu` argument specifies how the covariates enter into the model. It also fits an intercept as the first coefficient. So, the `pintercept` argument includes one value less than the number of intercepts fit. The intercept estimates below are a little off those estimated above.

```
nordr(w, dist="prop",mu=~ses+life,pmu=c(-.2,1,-.3), pintercept=c(1,2))   # see above for definition of w
```

```
Call:
nordr(w, dist = "prop", mu = ~ses + life, pmu = c(-0.5, 1, -0.3),
    pintercept = c(1, 2))

proportional odds model

-Log likelihood    49.54895
AIC                54.54895
Iterations         17

Location coefficients
Location function:
~ses + life
             estimate    s.e.
(Intercept)   -0.2817  0.6423
ses[.i]        1.1111  0.6109
life[.i]      -0.3189  0.1210

Intercept contrasts
      estimate    s.e.
b[2]     1.495  0.3898
b[3]     2.491  0.5012

Correlation matrix
        1       2       3       4       5
1  1.0000 -0.4449 -0.5924 -0.2557 -0.2229
2 -0.4449  1.0000 -0.1859  0.1885  0.2363
3 -0.5924 -0.1859  1.0000 -0.2312 -0.3003
4 -0.2557  0.1885 -0.2312  1.0000  0.7168
5 -0.2229  0.2363 -0.3003  0.7168  1.0000
```

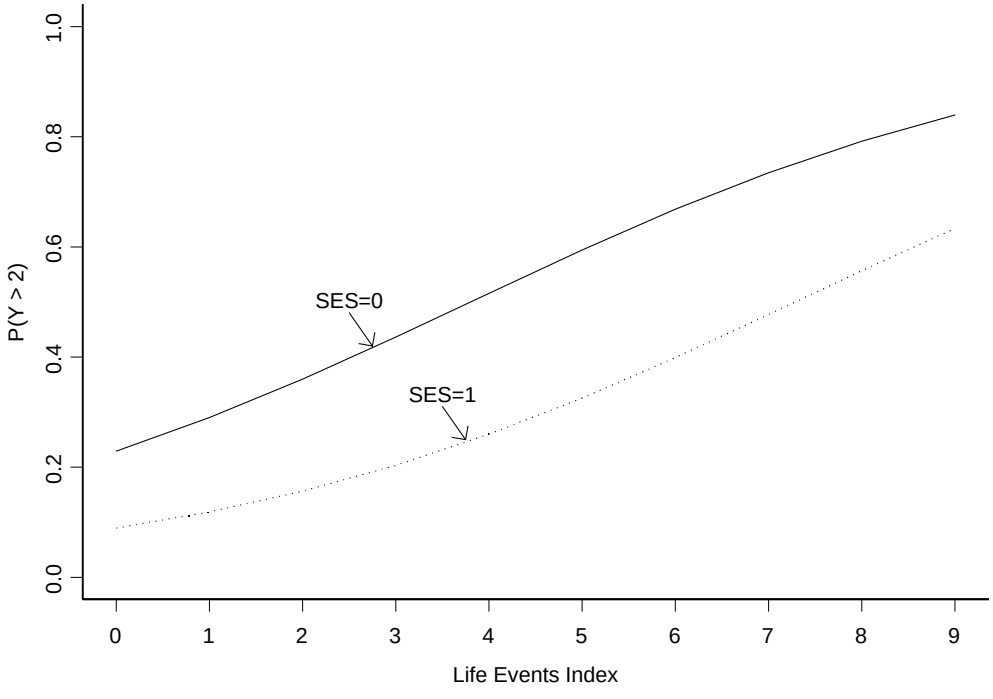
```
detach("table.7.5")
```

To obtain a score test, we extract the likelihood attribute from the fitted object `fit.lcr`, and compare it to the likelihood from fitting a model with separate sets of coefficients per response. This latter model can be conveniently fitted using `lcr` as well.

```
fit<-lcr(w,mu=~ses+life)   # no dist argument implies multinomial

2*(fit$likelihood-fit.lcr$likelihood)   # score test
[1] 2.399634

1-pchisq(2*(fit$likelihood-fit.lcr$likelihood),4)
[1] 0.6626934
```

To create Figure 7.6, we use the predicted probabilities from the `polr` object.

```
plot(0:9,rowSums(predict(fit.polr, newdata=data.frame(ses=rep(-1,10), life=-c(0:9)),
   type="probs")[,3:4]), axes=F,lty=2, type="l",ylim=c(0,1),bty="L",ylab="P(Y > 2)",
   xlab="Life Events Index")
axis(2)
axis(1, at=0:9)
lines(0:9,rowSums(predict(fit.polr, newdata=data.frame(ses=rep(0,10), life=-c(0:9)),
   type="probs")[,3:4]), lty=1)
text(2.5, .5, "SES=0")
arrows(2.5,.48,2.75,.42, open=T)
# R: arrows(2.5,.48,2.75,.42, length=.1)
text(3.5, .33, "SES=1")
arrows(3.5,.31,3.75,.25, open=T)
# R: arrows(3.5,.31,3.75,.25, length=.1)
```



We see that the probability of moderate or impaired mental health, P(Y > 2), is predicted to be greater with a higher life events index, and furthermore that a low SES has a higher predicted probability than higher SES.

Fitting interaction models is straightforward. Different link functions other than the logit link can be used in the function `lcr` and the function `vglm`, which is from the R package `VGAM` (coming soon to S-PLUS 6.x, but available for S-PLUS under L/Unix). This is illustrated in the next section. The function `vglm` can also fit partial proportional odds models mentioned on page 282 of Agresti. This is done by setting the `zero` argument of the family function, `cumulative`. See the help file for details.

D. Ordinal Responses: Cumulative Link Models

To illustrate the use of different link functions within an ordinal categorical model, Agresti uses a life table by race and sex (Table 7.7). The percentages within the four populations indicate that the underlying continuous cdf of life length increases slowly at low to moderate ages, then increases sharply at older ages. This pattern suggests a complementary log-log link function.

Several different link functions can be used within a proportional odds model. The function `lcr` in the R package `ordinal` fits a cumulative complementary log-log link model (as well as many other link functions).

First, we set up the data. I created a numeric response, `LifeC`, as well as a categorical one. `lcr` requires a numerical response.

```
table.7.7<-data.frame(expand.grid(Race=c(0,1),Sex=c(0,1), Life=ordered(c("0-20","20-
   40","40-50","50-60","over 65"))), LifeC=rep(0:4, each=4),
   percent=c(2.4,3.6,1.6,2.7, 3.4,7.5,1.4,2.9,3.8,8.3, 2.2,4.4,17.5,25,9.9, 16.3,
   72.9,55.6,84.9,73.7))
```

Then, we can use the utility functions mentioned above to allow easier fitting of the model.

```
library(ordinal)
y <- restovec(table.7.7$LifeC,weights=table.7.7$percent,type="ordinal") # response
    vector
tcc <- tcctomat(table.7.7$Race,name="Race")    # create covariate object
tcc <- tcctomat(table.7.7$Sex,name="Sex",oldccov=tcc)
w <- rmna(y,ccov=tcc)    # create a repeated object, with no repeats here
```

Finally, we attempt to use `lcr`. The argument `pcoef` gives starting values.

```
fit.lcr<-lcr(w,distribution="prop",mu=~Sex+Race,link="cloglog", pcoef=c(-3.7,-2.8,-
    2.2,-1.2,.658,-.626))
```

I have not been able to get convergence with this function.

The function `vglm` fits several multinomial models (including proportional odds and baseline-category) using IRLS. In fact, it is much easier to use than `lcr` for the purpose of fitting a proportional odds model. Here, we use it to fit the cumulative complementary log-log link model. If you don't use an ordered factor as the response, the function apparently assumes that the response categories appear consecutively in order.

```
library(vgam)
fit.vlgm<-vglm(Life~Sex+Race,family=cumulative(link=cloglog, parallel=T),
weights=percent, data=table.7.7)
```

In the above call we use the `cumulative` family with link `cloglog` and proportional odds (`parallel=T`).

```
summary(fit.vlgm)

Coefficients:
                Value Std. Error  t value
(Intercept):1 -3.73044    0.34277 -10.8832
(Intercept):2 -2.79987    0.24392 -11.4788
(Intercept):3 -2.21826    0.20688 -10.7227
(Intercept):4 -1.15449    0.16960  -6.8072
Sex           -0.65771    0.19566  -3.3614
Race           0.62641    0.19498   3.2127

Number of linear predictors:  4

Names of linear predictors:
cloglog(P[Y<=1]), cloglog(P[Y<=2]), cloglog(P[Y<=3]), cloglog(P[Y<=4])

Dispersion Parameter for cumulative family:   1

Residual Deviance: 699.7168 on 74 degrees of freedom

Log-likelihood: -349.8584 on 74 degrees of freedom

Number of Iterations: 6
```

Predictions are easy to get using the `predict` method. These are multiplied by 100 and rounded to the first decimal place.

```
t(round(predict(fit.vglm, type="response")[1:4,]*100,1))

          1    2    3    4
0-20    2.4  4.4  1.2  2.3
20-40   3.5  6.4  1.9  3.4
40-50   4.4  7.7  2.4  4.3
50-60  16.7 26.1  9.6 16.3
```

```
over 65 73.0 55.4 84.9 73.7
```

The function `vglm` will support dispersion effects, but it requires some work. In theory, it is possible to write your own family function to do this, by modifying the `cumulative` family.

E. Ordinal Responses: Adjacent-Categories Logit Models

Adjacent-categories logit models fit $J - 1$ logits involving the log odds of the probability that a response falls in category j given that it falls in either category j or $j + 1$, $j = 1,\ldots,J$. These logits are modeled as linear functions of explanatory variables as

$$\log\frac{\pi_j(\mathbf{x})}{\pi_{j+1}(\mathbf{x})} = \alpha_j + \boldsymbol{\beta}'\mathbf{x}, \qquad j = 1,\ldots,J-1 \tag{7.1}$$

Library functions exist for directly fitting these models in R. However, as Agresti shows on p. 287, they can also be fit using the equivalent baseline-category logit model. Thus, they can be fit in S-PLUS as well.

Agresti uses the Job Satisfaction Example to illustrate adjacent-categories logit models. The response variable is job satisfaction in categories (Very Dissatisfied, Little Satisfied, Moderately Satisfied, Very Satisfied). The explanatory variables are gender (1 = females) and income (< \$5,000, \$5,000 – \$15,000, \$15,000 – \$25,000, > \$25,000). Scores of 1 to 4 are used for income. The data set is available on Agresti's web site. Here we read it into R/S-PLUS and create an ordered factor out of the response.

```
table.7.8<-read.table("jobsat.r", col.names=c("gender","income","jobsat","freq"))
table.7.8$jobsatf<-ordered(table.7.8$jobsat, labels=c("very diss","little sat","mod
sat","very sat"))
```

To use `vglm` and the `acat` family (for adjacent categories), we need to modify the data set so that the responses are "unstacked". This is achieved as follows using the `unstack` function in R. (The `weights` argument for `vglm` is for inputting prior weights, not frequencies.)

```
table.7.8a<-
data.frame(expand.grid(income=1:4,gender=c(1,0)),unstack(table.7.8,freq~jobsatf))
# S-PLUS: table.7.8a<-data.frame(expand.grid(income=1:4,gender=c(1,0)),
menuUnstackColumns(source=table.7.8, source.col.spec=c("freq"),
group=c("jobsatf"),show.p=F)
```

```
 income gender very.diss little.sat mod.sat very.sat
1      1      1         1          3      11        2
2      2      1         2          3      17        3
3      3      1         0          1       8        5
4      4      1         0          2       4        2
5      1      0         1          1       2        1
6      2      0         0          3       5        1
7      3      0         0          0       7        3
8      4      0         0          1       9        6
```

Now, we fit the model using the cbinded responses on the left hand side. Note that the fit uses a model with negative the linear predictor $\boldsymbol{\beta}'\mathbf{x}$ in (7.1). So, the signs are different from Agresti's. To fix that, we could have negated gender and income prior to using them in the model. The `acat` family function has a `parallel` argument, which if true, fits proportional odds type model.

```
library(vgam)
```

```
fit.vglm<-vglm(cbind(very.diss,little.sat,mod.sat,very.sat)~gender+income,      family=
   acat(link="loge",parallel=T), data=table.7.8a)
summary(fit.vglm)

Coefficients:
                  Value Std. Error  t value
(Intercept):1  0.550668    0.67945  0.81046
(Intercept):2  0.655007    0.52527  1.24700
(Intercept):3 -2.025934    0.57581 -3.51842
gender        -0.044694    0.31444 -0.14214
income         0.388757    0.15465  2.51372

Number of linear predictors:  3

Names of linear predictors:
log(P[Y=2]/P[Y=1]), log(P[Y=3]/P[Y=2]), log(P[Y=4]/P[Y=3])

Dispersion Parameter for acat family:   1

Residual Deviance: 12.55018 on 19 degrees of freedom

Log-likelihood: -103.3293 on 19 degrees of freedom

Number of Iterations: 4
```

Because the coefficient on income is positive here, the odds of lower job satisfaction *decrease* as income increases.

F. Ordinal Responses: Continuation-Ratio Logit Models

Continuation-ratio logits are the logits of the probabilities $P(Y = j \mid Y \geq j)$ for response Y, $j = 1,\ldots,J-1$. These equal equation (7.12) in Agresti. Agresti shows that the likelihood of n is a product of multinomial mass functions which can in turn be factorized into products of binomial mass functions, using the equivalence between a joint probability mass function and the products of conditional probability mass functions. With different sets of parameters describing each of the $J - 1$ logits, maximization of the full likelihood can be done by maximizing each of the terms involving one of the continuation-ratio logits. These terms are products of binomial mass functions. Thus, maximum likelihood estimation can be carried out by fitting binomial logit models. However, using specialized software can make the estimation more efficient in practitioner-time.

Continuation-ratio logit models in R/S-PLUS can be fit using the function `glm`, `lrm` (in `Design` library), `nordr` (package `gnlm`), and `vglm` (package `vgam`). The last two are only present in R.

For this model, Agresti uses the data in Table 7.9 (p. 290). They come from a developmental toxicity study in pregnant mice. There were five concentration levels of the toxic substance (diEGdiME), one being a control. The response to the fetus was measured two days later and recorded as Nonlive, Malformed, or Normal. Continuation-ratio logits are used to model the probability of a nonlive fetus and the conditional probability of a malformed fetus, given that the fetus was live.

1. Using lrm from library Design

First, a continuation-ratio model is fit using `cr.setup` and `lrm` from the `Design` library.

It is important to add the libraries in this order:

```
library(Hmisc,T)
library(Design,T)
```

Now, we set up the data. I create two different data frames because we will use both later.

```
y<-ordered(c("non-live","malformed","normal"),levels=c("non-live","malformed",
   "normal"))
x<-c(0, 62.5, 125, 250, 500)
table.7.9<-data.frame(expand.grid(y=y, x=x), freq= c(15,1,281,17,0,225,
   22,7,283,38,59,202,144,132,9))
table.7.9a<-structure(.Data=menuUnstackColumns(source=table.7.9, source.col.spec=
   c("freq"), group=c("y"), show.p=F), row.names=x, names=unique(levels(y)))
# R: table.7.9a<-structure(.Data=unstack(table.7.9, freq~y),row.names=x)

     non-live malformed normal
   0       15         1    281
62.5       17         0    225
 125       22         7    283
 250       38        59    202
 500      144       132      9
```

To use cr.setup, we expand the response and x vectors according to their frequencies.

```
y<-rep(rep(y,5),table.7.9$freq)
x<-rep(x,tapply(table.7.9$freq, table.7.9$x, sum))
```

`cr.setup` will transform the response variable so that it can be used for a continuation-ratio model. In particular, it will create the new variables, `y` and `cohort`. The newly created variables are longer (have more observations) than the old response variable. `Cohort` defines the denominator categories for each logit (see equation (7.12) in Agresti). `y` is the transformed response variable taking on values 0 or 1 depending on whether the observation is a success or not within its cohort.

For example, for the data in Table 7.9, there are two cohorts. The first cohort ($j = 1$) is the set of all three categories: non-live, malformation, and normal, where an observation is considered a success if it falls in non-live versus either of the other two categories. The second cohort ($j = 2$) is the set of the last two categories, malformed and normal, where an observation is considered a success if it falls in malformed over normal.

Here is how to fit the model:

First set up the response:

```
u<-cr.setup(y)
y.u<-u$y
x.u<-x[u$subs]   # this ensures that the covariate is the correct length
```

I will do separate fits first before showing how to fit both models ($j = 1$ and $j = 2$) together. After the fit of the j = 1 model, I will discuss some of the output from `lrm` and compare it to Agresti's results and to results using `glm` and the other functions.

To fit the $j = 1$ model:

```
fit1<-lrm(y.u[u$cohort=="all"]~x.u[u$cohort=="all"])

Logistic Regression Model

Frequencies of Responses
    0   1
 1199 236

  Obs Max Deriv Model L.R. d.f. P     C   Dxy Gamma Tau-a    R2 Brier
 1435    2e-009     253.33    1 0 0.781 0.561 0.667 0.154 0.274 0.108

              Coef      S.E. Wald Z P
Intercept -3.247934 0.1576602 -20.6  0
      x.u  0.006389 0.0004348  14.7  0
```

The "Model L.R." given above is supposed to be the model likelihood ratio chisquare according to Harrell (1998). However, if you examine what `glm` gives, you can see that model L.R. is actually equal to –2LogLH(model with intercept + x). The d.f. above gives the number of d.f. used with the addition of x in the model (i.e., 1). What Agresti gives (p. 291) is the model residual deviance. That is, he gives –2LogLH(model with intercept only) – (–2LogLH(model with intercept + x). His d.f. correspond to the resulting d.f. when going from an intercept model (df = 4) to a model with x (df = 3). These are the df and LR statistic given directly by `glm` when modeling a linear logit, as shown later.

To fit the *j* = 2 model:

```
fit<-lrm(y.u[u$cohort=="y>=malformed"]~x.u[u$cohort=="y>=malformed"])

Logistic Regression Model

Frequencies of Responses
   0   1
1000 199

  Obs Max Deriv Model L.R. d.f. P     C   Dxy Gamma Tau-a    R2 Brier
 1199    5e-006     646.52    1 0 0.948 0.895  0.97 0.248 0.703 0.052

             Coef     S.E. Wald Z P
Intercept -5.70190 0.332249 -17.16 0
      x.u  0.01737 0.001227  14.16 0
```

See Harrell (1998) or the help file for the library for a full discussion of the other statistics produced by `lrm`.

To fit both models together (*j* = 1 and *j* = 2), fit an interaction term, as in the following.

```
fit<-lrm(y.u~u$cohort*x.u)

Logistic Regression Model

Frequencies of Responses
   0   1
2199 435

  Obs Max Deriv Model L.R. d.f. P     C   Dxy Gamma Tau-a    R2 Brier
 2634    4e-006     899.86    3 0 0.884 0.768 0.819 0.212 0.489 0.083

                           Coef      S.E. Wald Z P
             Intercept -3.247934 0.1576602 -20.60 0
   cohort=y>=malformed -2.453968 0.3677581  -6.67 0
                     x  0.006389 0.0004348  14.70 0
cohort=y>=malformed * x  0.010986 0.0013020   8.44 0
```

Thus, when y>=malformed (*j* = 2), the linear logit is –5.70 + .017*x*. When y=all (*j* = 1), the linear logit is –3.247 + .0064*x*. The less desirable outcome is more likely as the concentration increases.

Notice that the values for model L.R. in the individual model sum to the model L.R. for the interaction model above. However, the d.f. do not add.

1. Odds Ratios

To get selected odds ratios for the *j* = 2 model, first issue the `datadist` command and reissue the `lrm` call, as follows:

```
x.u<-x.u[u$cohort=="y>=malformed"]
dd<-datadist(x.u)
options(datadist='dd')
fit<-lrm(y.u[u$cohort=="y>=malformed"]~x.u)
```

Using `summary(fit)` will give odds ratios comparing the default levels of `x.u` (the lowest and highest nonzero values)

```
summary(fit)
              Effects              Response : y.u[u$cohort == "y>=malformed"]

      Factor  Low High Diff. Effect S.E. Lower 0.95 Upper 0.95
 x.u         62.5 250  187.5  3.26  0.23  2.81        3.71
  Odds Ratio 62.5 250  187.5 25.99    NA 16.56       40.80
```

Thus, given that a fetus was alive, the estimated odds of it being malformed versus normal is 26 times higher when a mouse is exposed to 500 mg/kg per day of the toxic substance than when it is exposed to 62.5 mg/kg per day. The NA for SE is apparently not a mistake. Also the value 3.26 = (.0174)*(250-62.5) is the log odds.

To get an odds ratio comparing specific levels of x, for example comparing levels x=125 and x=250:

```
summary(fit,x=c(125,250))

              Effects              Response : y.u[u$cohort == "y>=malformed"]

      Factor Low High Diff. Effect S.E. Lower 0.95 Upper 0.95
 x.u         125 250  125   2.17   0.15 1.87        2.47
  Odds Ratio 125 250  125   8.77     NA 6.50       11.85
```

Or, levels x=250 and x=500

```
summary(fit,x=c(250,500))

              Effects              Response : y.u[u$cohort == "y>=malformed"]

      Factor Low High Diff. Effect S.E. Lower 0.95 Upper 0.95
 x.u         250 500  250    4.34  0.31  3.74        4.95
  Odds Ratio 250 500  250   76.99    NA 42.20      140.48
```

2. Using glm

We can instead estimate the continuation-ratio logit model using `glm`, as we did the linear logit model.

First, set x (the covariate) and the weights for the first linear logit model. Here, we use the `table.7.9a` version of the data.

```
x<-c(0,62.5,125,250,500)
n1<-rowSums(table.7.9a) # use the whole table
```

For model *j* = 1, take the first column of `table.7.9a` as the "success" , as follows:

```
(fit<-glm(table.7.9a[,1]/n1~x, family=binomial,weights=n1))

Coefficients:
 (Intercept)           x
   -3.247934 0.006389069

Degrees of Freedom: 5 Total; 3 Residual
Residual Deviance: 5.777478
```

```
fit$null.deviance
[1] 259.1073

fit$null.deviance-fit$deviance    # this is what lrm gave us as model L.R.
[1] 253.3298

summary(fit)

    Null Deviance: 259.1073 on 4 degrees of freedom

Residual Deviance: 5.777478 on 3 degrees of freedom
```

The difference of the above sets of values gives Null Deviance-Residual Deviance = 253.3298 and df=1. These are the Model L.R. and Model df reported by `lrm`.

For the *j* = 2 model, take the second and third columns of `table.7.9a`, and use the second column as the success:

```
n2<-rowSums(table.7.9a[,c(2,3)])

glm(table.7.9a[,2]/n2~x, family=binomial,weights=n2)

Coefficients:
 (Intercept)           x
   -5.701891 0.01737464

Degrees of Freedom: 5 Total; 3 Residual
Residual Deviance: 6.060908
```

`glm` can also fit proportional odds models. However, there are advantages to using `lrm` or other functions for these types of models because of the built-in features, like the odds ratios above. See Harrell(1998) or the associated web site http://hesweb1.med.virginia.edu/biostat (under Statistical Computing Tools) for more information.

3. Using nordr from library gnlm and lcr from library ordinal

These two functions are very similar. However, they assume a common concentration slope across continuation-ratio logits. I will illustrate `lcr`. First, it becomes easier to use the function if we create a `repeated` object as follows. The response factor must be transformed to a numeric vector, starting with response 0. Then, we create response and covariate objects, followed by a `repeated` object using `rmna`.

```
table.7.9$yC<-codes(table.7.9$y)-1 # must transform to 0:2 categories
y <- restovec(table.7.9$yC,times=F,weights=table.7.9$freq,type="ordinal")  # response vector
tcc <- tcctomat(table.7.9$x,name="Concen")    # create covariate object
w <- rmna(y,ccov=tcc)    # create a repeated object, with no repeats here
```

The model is fit by

```
library(ordinal)
lcr(w,distribution="cont",direc="up",mu=~Concen)

Frequency table.
Number of non-empty cells: 15
Total number of events:    1435

Continuation-ratio distribution (upwards).
Transformation: identity.

Regression coefficients
              estimate       s.e.
(Intercept0)  -2.632877  0.1825725
```

```
(Intercept1)  -2.404463  0.1078119
Concen         0.007184  0.0003578

Correlation matrix
         1        2        3
1   1.0000   0.6162  -0.7963
2   0.6162   1.0000  -0.7738
3  -0.7963  -0.7738   1.0000
```

Note the different estimated concentraton effect.

4. Using vglm from library VGAM

Using `vglm` for continuation-ratio models is similar to using it for adjacent-categories logit models. We just change the family function to `sratio` (or `cratio`). First, I illustrate `sratio`, as this matches Agresti's definition of continuation-ratio logit. Both `sratio` and `cratio` have an argument, `reverse`, which fits the corresponding logits in reverse order (see the help files).

For the data, we use the contingency table format (`table.7.9a`), and `cbind` the response columns.

```
x<-c(0,62.5,125,250,500)
fit.vglm<-vglm(cbind( non.live, malformed,normal )~x,family=sratio(link ="logit",
parallel = F), data=table.7.9a)
summary(fit.vglm)

Coefficients:
                   Value Std. Error t value
(Intercept):1 -3.2479337 0.15766019 -20.601
(Intercept):2 -5.7018965 0.33062798 -17.246
x:1            0.0063891 0.00043476  14.695
x:2            0.0173747 0.00121260  14.328

Number of linear predictors:  2

Names of linear predictors: logit(P[Y=1|Y>=1]), logit(P[Y=2|Y>=2])

Dispersion Parameter for sratio family:   1

Residual Deviance: 11.83839 on 6 degrees of freedom

Log-likelihood: -730.3872 on 6 degrees of freedom

Number of Iterations: 4
```

Note that the Residual Deviance reported is the sum of the individual Residual Deviances (the likelihood-ratio fit statistics) reported by `glm`.

Now, the family function cratio fits the logits, logit(P(y > j | y $\geq$ j), which is not the same as logit(P(y = j | y $\geq$ j). In fact it is the logit of the complement, under the conditioning. Here is that fit.

```
x<-c(0,62.5,125,250,500)   # this is the form of the covariate we use
fit.vglm<-vglm(cbind(non.live, malformed, normal)~x, family=cratio(link ="logit",
parallel = F), data=table.7.9a)
summary(fit.vglm)

Log-likelihood: -730.3871
Coefficients:
                   Value Std. Error t value
(Intercept):1  3.2479337 0.15766019  20.601
(Intercept):2  5.7018965 0.33062798  17.246
x:1           -0.0063891 0.00043476 -14.695
x:2           -0.0173747 0.00121260 -14.328
```

```
Number of linear predictors:  2

Names of linear predictors: logit(P[Y>1|Y>=1]), logit(P[Y>2|Y>=2])

Dispersion Parameter for cratio family:   1

Residual Deviance: 11.83839 on 6 degrees of freedom

Log-likelihood: -730.3872 on 6 degrees of freedom

Number of Iterations: 4
```

It is not surprising to see that the estimates are all negated, as we are fitting the logits of the complementary probabilities.

As a side note, as mentioned in Harrell (1998) as well as Agresti (Chapter 7 notes), the continuation-ratio model is a discrete version of the Cox proportional hazards model. Thus, one could probably fit these models using either `coxph` or `cph`, which is in the Design library. It is left to the reader to make the connection.

G. Ordinal Responses: Mean Response Models

A mean response model for ordinal responses is a linear regression model with ordinal responses represented by numerical scores. As in ordinary linear regression with continuous response, the conditional mean is assumed linearly related to the explanatory variables. The response distribution is assumed product multinomial (i.e., independent multinomial at each fixed set of covariates).

This type of model can be fit using something like the `lpmreg` function of Chapter 4.

Also, using the Poisson/multinomial connection, we might try `glm` with a `poisson(identity)` family. Or, we might try `aov`, as we are fitting a linear regression model. The functions `glm` and `aov` give similar coefficient estimates. However, they are not exactly the same as those assuming the product-multinomial model. Also, the standard errors are wrong.

```
fit.aov<-aov(jobsat ~ gender + income, data = table.7.8, weights = freq)
fit.aov$coefficients
```

```
(Intercept)      gender      income
 2.57391211 -0.01703921  0.17985911
```

```
fit.glm<-glm(jobsat~gender+income, weights=freq, family=poisson(identity), data=
table.7.8)
summary(fit.glm)
```

```
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  2.561845   0.557081   4.599 4.25e-06 ***
gender      -0.007199   0.370796  -0.019    0.985
income       0.182282   0.169106   1.078    0.281
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 19.435  on 25  degrees of freedom
Residual deviance: 18.124  on 23  degrees of freedom
AIC: 332.31

Number of Fisher Scoring iterations: 3
```

Multinomial-Poisson Homogeneous Model

A better alternative for fitting mean response models using ML estimation is to use J. Lang's `mph.fit` function for R, which fits Multinomial-Poisson Homogenous models. For the mean response model, we assume product-multinomial sampling, given the totals from the eight populations.

To use the function `mph.fit`, we need Table 7.8 in the follow form, as a 8 x 4 matrix of counts.

```
table.7.8a<-data.frame(expand.grid(income=1:4,gender=c(1,0)),
   unstack(table.7.8,freq~jobsatf))
table.7.8a<-structure(.Data=table.7.8a[,-(1:2)], row.names= apply(expand.grid(
   income=1:4, gender=c(1,0)), 1, paste,collapse=" "), names=
   levels(table.7.8$jobsatf) )
```

Because this manual was originally written when version 1.0 of `mph.fit` was available, I give a description of its use, along with the most recent version 3.1. If a command is specific to a particular version, I note that.

For version 1.0 of `mph.fit`, we define two sampling matrices, Z and ZF. Z is a 32 x 8 matrix, where each column corresponds to a stratum or population (of which we have 8). If each response is possible within each stratum, then the Z matrix (which is called a *population matrix*) can be generated using the function `pop`, which creates a block diagonal matrix

```
Z<-pop(npop=8, nlev=4) # 8 populations, 4 levels of response
# same as Z<-kronecker(diag(8), matrix(1,4,1))
```

ZF is a sampling constraint matrix, whose columns tell whether each of the row totals is fixed or random (Poisson distributed). If the *j*th column of Z is included in ZF, then the *j*th population total is fixed.

```
ZF<-Z
```

Next, we need the vector of table counts and the design matrix, X

```
# vector of table counts
Y<-c(t(as.matrix(table.7.8a)))

# design matrix
X<-cbind(1,unique(as.matrix(table.7.8[,1:2])))
X<-as.data.frame(X)

model.matrix(~gender+income,data=X)
```

```
   (Intercept) gender income
1            1      1      1
5            1      1      2
9            1      1      3
13           1      1      4
17           1      0      1
21           1      0      2
25           1      0      3
29           1      0      4
attr(,"assign")
[1] 0 1 2
```

Next, the matrix `A` and the result of the function `L.fct` together define the formula that describes the dependence of the mean response on the covariates. See the `mph` documentation for an explanation of `L.fct`.

```
scores<-1:4
```

```
A<-kronecker(diag(8),matrix(scores,1,4))

L.fct <- function(m) {
  p <- diag(c(1/(Z%*%t(Z)%*%m)))%*%m
  A<-kronecker(diag(8),matrix(scores,1,4))
  A%*%p
}
```

Finally, we fit a mean response model. For version 3.1, the argument `L.mean=T` indicates that `L.fct` is a function of the expected counts, and not of the table probabilities. The argument `strata` indicates which count belongs to which of the eight strata, and `fixed.strata="all"` indicates that all stratum totals are fixed. These arguments help to create the Z and ZF matrices within version 3.1 of `mph.fit`.

```
# fit<-mph.fit(y=Y,Z=Z,ZF=ZF,X=X,L.fct=L.fct) # version 1.0
fit<-mph.fit(y=Y, X=model.matrix(~gender+income,data=X), L.fct=L.fct, L.mean=T,
strata=rep(1:8, each=4), fixed.strata="all",  maxiter=500)
mph.summary(fit)

MODEL GOODNESS OF FIT:   Test of   Ho: h(m)=0 vs. Ha: not Ho...

  Likelihood Ratio Stat (df= 5 ):  Gsq =  5.06161 (pval =  0.4084 )
  Pearson's Score Stat  (df= 5 ):  Xsq =  4.64931 (pval =  0.4602 )
  Generalized Wald Stat (df= 5 ):  Wsq =  5.16351 (pval =  0.3963 )

  WARNING: 78.125% of expected counts are less than 5.
           Chi-square approximation may be questionable.

  Adj Resids: -1.436 -1.355 ... 1.436 1.436 , Number |Adj Resid| > 2:  0

SAMPLING PLAN INFORMATION...
Number of strata:   8
Strata identifiers: 1, 2, 3, 4, 5, 6, 7, 8
Strata with fixed sample sizes: all
Observed strata sample sizes:   17, 25, 14, 8, 5, 9, 10, 16

LINEAR PREDICTOR MODEL RESULTS...
              BETA StdErr(BETA) Z-ratio   p-value
(Intercept)  2.5927       0.2408 10.7685 0.0000000
gender      -0.0298       0.1449 -0.2056 0.8371200
income       0.1807       0.0694  2.6027 0.0092496

      OBS LINK ML LINK StdErr(L) LINK RESID
link1   2.8235  2.7436    0.1237     0.6151
link2   2.8400  2.9242    0.0882    -0.7813
link3   3.2857  3.1049    0.0996     1.3546
link4   3.0000  3.2856    0.1472    -1.4359
link5   2.6000  2.7734    0.1798    -0.4259
link6   2.7778  2.9540    0.1278    -0.9911
link7   3.3000  3.1347    0.0999     1.0393
link8   3.3125  3.3154    0.1152    -0.0320

CONVERGENCE INFORMATION...
  Original counts used.
  iterations = 288 ,  time elapsed = 9.03
  norm.diff  = 3.15141 = dist between last and second last iterates.
               Did NOT meet norm diff convergence criterion [1e-06]!
  norm.score = 8.22404e-07 = norm of score at last iteration.
               Norm score convergence criterion [1e-06] was met.

FITTING PROGRAM USED:  mph.fit, version 3.1, 5/20/09
```

Note that only one of the convergence criteria was met.

One can modify the `mph.fit` function to work with S-PLUS. To do so, one must take account of the differences within the scoping rules of the two implementations.

Weighted Least Squares

It is possible to work through a WLS estimation in either S-PLUS or R using the formula on p. 600 – 604 of Agresti. We do this for the job satisfaction data.

First, I reorganize the table of data by unstacking the `jobsat` variable. The `menuUnstackColumns` function in S-PLUS unstacks `jobsat` and creates a new data frame called `table.7.8a`. The `unstack` function in R works similarly, but is easier to use.

```
menuUnstackColumns(source=as.data.frame(table.7.8), target=table.7.8a,
   source.col.spec= c("freq"), group=c("jobsat"), show.p=F)
# R: table.7.8a<-data.frame(expand.grid(income=1:4,gender=c(1,0)),
   unstack(table.7.8,freq~jobsatf))
structure(.Data=table.7.8a,row.names=apply(expand.grid(income=1:4,gender=c(1,0)),1,
   paste,collapse=" "), names=levels(table.7.8$jobsatf) )
# R: structure(.Data=table.7.8a[,-(1:2)],row.names=apply(expand.grid(income=1:4,
   gender=c(1,0)),1, paste,collapse=" "), names=levels(table.7.8$jobsatf) )

    very diss little sat mod sat very sat
1 1         1          3      11        2
2 1         2          3      17        3
3 1         0          1       8        5
4 1         0          2       4        2
1 0         1          1       2        1
2 0         0          3       5        1
3 0         0          0       7        3
4 0         0          1       9        6
```

Next, we need the sample size for each population (income x gender combination), and the sample proportions for each population.

```
n<-rowSums(table.7.8a)

p<-sweep(table.7.8a,1,n,FUN="/")
p1<-p[1,] # R: as.numeric(p[1,])
p2<-p[2,]
p3<-p[3,]
p4<-p[4,]
p5<-p[5,]
p6<-p[6,]
p7<-p[7,]
p8<-p[8,]

J<-4   # number of response categories
I<-8   # number of populations
```

Now, I compute the block diagonal variance-covariance matrix, $\mathbf{V}$, using $\mathbf{V}_1$ through $\mathbf{V}_8$ (the population specific covariance matrices). For the for-loop below, the object `Vnames` stores the names of the objects $\mathbf{V}_1$ through $\mathbf{V}_8$, and the object `pnames` stores the names of the vectors of sample proportions for each group. First I calculate each $\mathbf{V}_i$, then I calculate $\mathbf{V}$.

```
Vnames<-sapply(1:I,function(x) paste("V", x, collapse = " ", sep = ""))
pnames<-sapply(1:I,function(x) paste("p",x,collapse=" ",sep=""))
V<-matrix(0,nc=J,nr=J)

for(i in 1:I)
{
   p<-as.numeric(eval(parse(text = pnames[i])))
```

```
    diag(V)<-p*(1-p)
    p<-as.matrix(p)
    junk<-matrix(-kronecker(p,p),nc=J,nr=J,byrow=T)
    V[lower.tri(diag(J))]<-junk[lower.tri(junk)]
    V<-t(V)
    V[lower.tri(V)]<-junk[lower.tri(junk,diag=F)]
    assign(Vnames[i], matrix(V/n[i], ncol = J, byrow = T))
}
```

I construct **V** using $\mathbf{V}_1$ through $\mathbf{V}_8$

```
zero<-matrix(0,J,J)
V<-rbind(
cbind(V1, zero, zero, zero, zero, zero, zero, zero),
cbind(zero, V2, zero, zero, zero, zero, zero, zero),
cbind(zero, zero, V3, zero, zero, zero, zero, zero),     # block-diagonal matrix, V
cbind(zero, zero, zero, V4, zero, zero, zero, zero),
cbind(zero, zero, zero, zero, V5, zero, zero, zero),
cbind(zero, zero, zero, zero, zero, V6, zero, zero),
cbind(zero, zero, zero, zero, zero, zero ,V7, zero),
cbind(zero, zero, zero, zero, zero, zero, zero, V8)
)
```

Now, the model for this example has form $F(\boldsymbol{\pi}) = \mathbf{X}\boldsymbol{\beta}$, where $\boldsymbol{\pi}$ is a 4 x 8 matrix of response probability distributions for each population, $\boldsymbol{\beta} = (\alpha, \beta_x, \beta_g)^{\mathrm{T}}$, and $F(\boldsymbol{\pi})$ is a 8 x 1 vector of the 8 response functions

$$F_1(\boldsymbol{\pi}) = \alpha + 0 \cdot \beta_g + 1 \cdot \beta_x = \boldsymbol{\nu}^{\mathrm{T}}(\pi_{1(1,1)},...,\pi_{4(1,1)})$$
$$\vdots$$
$$F_8(\boldsymbol{\pi}) = \alpha + \beta_g + 4 \cdot \beta_x = \boldsymbol{\nu}^{\mathrm{T}}(\pi_{1(2,4)},...,\pi_{4(2,4)})$$

representing the 2 x 4 combinations of Gender and Income. The 4 x 1 vector $\boldsymbol{\nu}$ contains the scores for the 4 job satisfaction categories.

Now, note that the matrix $\mathbf{Q} = \left[\frac{\partial F_k(\boldsymbol{\pi})}{\partial \pi_{j|i}}\right]$ on p. 602 contains the 1 x 32 row vectors

$$\frac{\partial F_k(\boldsymbol{\pi})}{\partial \pi_{j|i}} = (0_1,...,0_{(k-1)4},\nu_1,...,\nu_4,0,...,0_{32}),$$

and the design matrix is

```
   ..1 gender income
    1      1      1
    1      1      2
    1      1      3
    1      1      4
    1      0      1
    1      0      2
    1      0      3
    1      0      4
```

All of the above is input into S via the following commands. For the F( ) functions, we use the sample proportions instead of the probabilities, $\boldsymbol{\pi}$.

```
js<-1:4   # the vector nu

Q<-rbind(
   c(js,rep(0,28)),
```

```
    c(rep(0,4),js,rep(0,24)),
    c(rep(0,8),js,rep(0,20)),
    c(rep(0,12),js,rep(0,16)),  # derivatives of F
    c(rep(0,16),js,rep(0,12)),
    c(rep(0,20),js, rep(0,8)),
    c(rep(0,24),js, rep(0,4)),
    c(rep(0,28),js)
)


VF<-Q%*%V%*%t(Q)    # transformed covariance matrix (p. 602 in Agresti)

# Design matrix:
X<-as.matrix(cbind(rep(1,I),unique(table.7.8[,1:2])))

# Functions
Fp<-c(js%*%p1, js%*%p2, js%*%p3, js%*%p4, js%*%p5, js%*%p6, js%*%p7, js%*%p8)
```

Now, I estimate beta using the formula in Section 15.1.2, the weighted least squares estimator.

```
InvVF<-solve(VF)
Covb<-solve(t(X)%*%InvVF%*%X)
b<-as.numeric(Covb%*%t(X)%*%InvVF%*%Fp)
[1]  2.61328732 -0.04155966  0.18163655
```

The asymptotic standard errors:

```
sqrt(diag(Covb))
[1] 0.2311374 0.1369473 0.0681758
```

I compute a Wald statistic for entire model using the formula on p. 603 in Agresti.

```
as.numeric(t(Fp-X%*%b)%*%InvVF%*%(Fp-X%*%b))
 5.242789
```

H. Generalized Cochran-Mantel Haenszel Statistic for Ordinal Categories

The `mantelhaen.test` function in the R package `ctest`, which comes with R, can handle *I* x *J* x *K* tables, but does not take special advantage of ordinal categories (The same built-in function in S-PLUS can only handle 2 x 2 x 2 tables, but the statistic itself is easily calculable using formula that is already programmed into the R function. A calculation is available in the S-PLUS scripts for this manual). To test for conditional independence of job satisfaction and income given gender, treating job satisfaction and income with scores {1, 3, 4, 5} and {3, 10, 20, 35}, respectively, we need to use equation (7.21) in Agresti. This statistic has an approximate chi-squared distribution with 1 df.

With nominal categories for job satisfaction and income, `mantelhaen.test` calculates (7.20) in Agresti using input from `xtabs`.

```
mantelhaen.test(xtabs(freq~jobsatf+income+gender, data=table.7.8))

        Cochran-Mantel-Haenszel test

data:  xtabs(freq ~ jobsatf + income + gender, data = table.7.8)
Cochran-Mantel-Haenszel M^2 = 12.5314, df = 9, p-value = 0.185
```

But, this gives a different answer than that given by SAS in Table 7.12 of Agresti. I don't know why, as it is clear from the R code that it is computing (7.20).

The test of nonzero correlation uses the ordinal scores for both row and column variables and computes (7.21). This computation is easy in either R or S-PLUS.

First, we add the scores to the data frame

```
table.7.8$jobsatS<-ifelse(table.7.8$jobsat==4,5,table.7.8$jobsat)
table.7.8$jobsatS<-ifelse(table.7.8$jobsat==3,4,table.7.8$jobsatS)
table.7.8$jobsatS<-ifelse(table.7.8$jobsat==2,3,table.7.8$jobsatS)

table.7.8$incomeS<-ifelse(table.7.8$income==4,35,table.7.8$income)
table.7.8$incomeS<-ifelse(table.7.8$income==3,20,table.7.8$incomeS)
table.7.8$incomeS<-ifelse(table.7.8$income==2,10,table.7.8$incomeS)
table.7.8$incomeS<-ifelse(table.7.8$income==1,3,table.7.8$incomeS)
```

Then, we compute T_k, its expected value under no correlation and its variance.

```
table.7.8.array<-xtabs(freq~jobsatf+income+gender, data=table.7.8)
Tk<-apply(table.7.8.array,3,function(x,u,v) sum(outer(u,v)*x), u=c(1,3,4,5),
   v=c(3,10,20,35))
ETk<-apply(table.7.8.array,3,function(x,u,v)
   sum(rowSums(x)*u)*sum(colSums(x)*v)/sum(x), u=c(1,3,4,5), v=c(3,10,20,35))

varTk<-apply(table.7.8.array,3,function(x,u,v) {
    n<-sum(x)
    rowsums<-rowSums(x)
    colsums<-colSums(x)
    (sum(rowsums*u^2) - (sum(rowsums*u)^2)/n)*(sum(colsums*v^2) -
   (sum(colsums*v)^2)/n)/(n-1)
}, u=c(1,3,4,5), v=c(3,10,20,35))
```

The statistic is

```
(sum(Tk-ETk)^2)/sum(varTk)
[1] 6.156301
```

with p-value

```
1-pchisq((sum(Tk-ETk)^2)/sum(varTk),df=1)
[1] 0.01309447
```

The *row mean scores differ* association treats rows as nominal and columns as ordinal. The test statistic can be computed in R or S-PLUS using the formula in the notes for Chapter 7 (p. 302, Agresti). However, the matrices in this formula do not have the appropriate dimensions for multiplication. So, one may have to check the original paper. In any case, the `mantelhaen.test` function can be modified easily to incorporate the **B** matrix given on p. 302, to get the new statistic.

Chapter 8 –Loglinear Models for Contingency Tables

A. Summary of Chapter 8, Agresti

Loglinear models are used for modeling cell counts in a contingency table. As Agresti notes, these models are usually used when we have a multivariate response, not a univariate response, as the model treats all classification factors as responses. With a univariate response, methods such as logit models or multinomial logit models are better alternatives, and there is a correspondence between logit models with categorical explanatory variables and loglinear models (Section 8.5, Agresti). Loglinear models model the expected cell frequencies as log linear combinations of effects (model parameters) due to each classification factor by itself and possibly due to interactions among classification factors. Certain contrasts involving parameters (such as differences between parameters for a given factor) are interpreted as log odds of making one response on that variable, relative to another response. Contrasts involving interaction parameters can have interpretations in terms of log odds ratios. Interaction parameters by themselves are most useful in an association interpretation. That is, a three-factor interaction parameter is zero if there is no three-factor association. For identifiability of all parameters, arbitrary constraints are made. This makes the individual parameter estimates not unique across constraints, but estimated contrasts that encode log odds ratios are the same across constraints.

A generalized linear model interpretation of loglinear models treats the $N = IJ$ cell counts of an I x J table as independent observations from a Poisson random component with corresponding means equal to the expected cell counts. The same model can have a multinomial interpretation with two categorical responses and N total observations. These statements also apply to three-way tables.

In a three-way table with response variables X, Y, and Z, several types of potential independence can be present. *Mutual independence* of all three variables results in all interaction parameters in the loglinear model being zero. In a multinomial interpretation, this means all joint cell probabilities equal the products of the corresponding marginal probabilities. *Joint independence* of one variable, Y, and the combined classifications of the other two (X and Z) results in a loglinear model with only one possible nonzero interaction parameter, that between X and Z. Finally, variables X and Y are *conditionally independent* of variable Z if independence holds within each partial table conditional on a given value of Z. This would result in a loglinear model with two possible nonzero interaction parameters: that describing association between X and Z and that describing association between Y and Z. Relationships among the types of independence appear in Table 8.1. in Agresti.

A loglinear model for no three-factor interaction in a three-way table is called a *homogenous association* model. This means that the conditional odds ratios between any two variables are identical at each category of the third variable. Its parameters have interpretations in terms of conditional odds ratios. Of the ${}_{I}P_{2} \times {}_{J}P_{2}$ possible odds ratios, there are $(I - 1)(J - 1)$ nonredundant odds ratios describing the association between variables X and Y, at each of K levels of a third conditioning variable. Conditional independence of X and Y (i.e., no three-factor interaction) means that all of the odds ratios are equal to 1.0. The logs of these odd ratios are functions of the parameters of the homogeneous association model, as shown in equation (8.14) in Agresti, and the functions do not depend on the level of the conditioning variable.

Higher dimensional tables have straightforward extensions from the three-way table.

Chi-squared goodness-of-fit tests can be used to compare nested models. This usually means using the likelihood ratio chi-squared statistic, which has an asymptotic chi-squared distribution when the expected frequencies are large (with fixed number of cells). The degrees of freedom equal the difference in dimension between the null and alternative hypothesis.

Loglinear models can be fit using one of two methods for doing maximum likelihood estimation, Newton-Raphson or Iterative proportional fitting (IPF). Newton-Raphson is an iterative procedure that solves a weighted least squares equation at each iteration. At each iteration of the IPF algorithm, the fitted values satisfy the model and eventually converge to match the sufficient statistics (leading to MLEs). However, IPF does not automatically produce the estimated covariance matrix of the parameters as a byproduct. ML parameter estimates have asymptotic normal distributions and asymptotic standard errors estimated by the inverse of the Information matrix of the log likelihood.

A generalized loglinear model generalizes mean modeling for loglinear models such as Poisson GLIMs. Specifically, the link function relating the response mean to the linear predictor is not strictly logarithmic and not necessarily invertible, and takes the form

$$\mathbf{C}(\log \mathbf{A}\boldsymbol{\mu}) = \mathbf{X}\boldsymbol{\beta}$$

for matrices **C** and **A**, which are not necessarily invertible. With **A** and **C** identity matrices, we get ordinary loglinear models. For example, the logit model for a three-way table that postulates independence of the response and the two explanatory variables, each with two levels (i.e., the logit model only has an intercept) has **A** equal to an 8 x 8 identity matrix so that $\boldsymbol{\mu} = (\mu_{111}, \mu_{112}, ..., \mu_{222})$ and

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix}$$

With **X** a vector of four ones, and $\boldsymbol{\beta} = \alpha$, we get four row logits (two per each level of the second explanatory variable) that are all equal.

B. Loglinear Models for Three-way Tables

With a three-way table, there are three response variables for a loglinear model. We can test the fit of a saturated model (three-way interaction) against a homogeneous association model (all pairwise associations) or that of homogeneous association versus conditional independence of two variables given a third. Agresti uses a data set on alcohol, cigarette, and marijuana use to fit these various models. We can set up the data as

```
table.8.3<-data.frame(expand.grid(
   marijuana=factor(c("Yes","No"),levels=c("No","Yes")),
   cigarette=factor(c("Yes","No"),levels=c("No","Yes")),
   alcohol=factor(c("Yes","No"),levels=c("No","Yes"))),
   count=c(911,538,44,456,3,43,2,279))
```

Fitting a loglinear model can be done using Iterative Proportional Fitting (`loglin`, `loglm`) or Newton Raphson (`glm` with poisson family). The former uses `loglin`, or `loglm` from MASS. The function `loglm` is a front-end for `loglin`, and has a much more flexible input allowance. `loglin` requires input in the form of output from `table()` or as an array. `loglm` accepts `table` output, `crosstabs` output (`xtabs` in R), or a formula using variables from a data frame. As we have a data frame, we start with `loglm`. We fit the saturated model, homogeneous association model, etc down to the marginal independence model.

Here are the fits. I set the arguments `fit` and `param` to `T` so that I can get fitted values and parameter estimates, as we see later.

```
library(MASS)

fitACM<-loglm(count~alcohol*cigarette*marijuana,data=table.8.3,param=T,fit=T)      # ACM
fitAC.AM.CM<-update(fitACM, .~. - alcohol:cigarette:marijuana)                     # AC, AM, CM
fitAM.CM<-update(fitAC.AM.CM, .~. - alcohol:cigarette)                             # AM, CM
fitAC.M<-update(fitAC.AM.CM, .~. - alcohol:marijuana - cigarette:marijuana)        # AC, M
fitA.C.M<-update(fitAC.M, .~. - alcohol:cigarette)                                 # A, C, M
```

We can get the fitted counts using `fitted` on the `loglm` objects. `fitted` returns an array of expect counts, but here we want a vector. I needed to transpose before concatenation so that the expected counts followed that in the data frame.

```
data.frame(table.8.3[,-4], ACM=c(aperm(fitted(fitACM))),
   AC.AM.CM=c(aperm(fitted(fitAC.AM.CM))), AM.CM=c(aperm(fitted(fitAM.CM))),
   AC.M=c(aperm(fitted(fitAC.M))), A.C.M=c(aperm(fitted(fitA.C.M))))

  marijuana cigarette alcohol ACM   AC.AM.CM       AM.CM      AC.M     A.C.M
1       Yes       Yes     Yes 911 910.383057 909.2395630 611.17749 539.98254
2        No       Yes     Yes 538 538.616089 438.8404236 837.82251 740.22607
3       Yes        No     Yes  44  44.616840  45.7604179 210.89632 282.09125
4        No        No     Yes 456 455.385590 555.1595459 289.10370 386.70007
5       Yes       Yes      No   3   3.616919   4.7604165  19.40246  90.59739
6        No       Yes      No  43  42.383881 142.1595764  26.59754 124.19392
7       Yes        No      No   2   1.383159   0.2395833 118.52373  47.32881
8        No        No      No 279 279.614380 179.8404236 162.47627  64.87991
```

Because we set `param = T`, we can get estimates of the model parameters (the lambdas), using `fit$param`. These can be used to get estimates of conditonal odds ratios, as per equation (8.14) in Agresti (see Thompson (1999, p. 27)). However, it may be simpler to just compute the odds ratios from the array of fitted values. To do this, we use the `apply` function on the array. For example, for the homogeneous association model (AC, AM, CM), which has fitted counts close to the observed counts, we have the following conditional odds ratios:

```
fit.array<-fitted(fitAC.AM.CM)

odds.ratio<-function(x) x[1,1]*x[2,2]/(x[2,1]*x[1,2])

apply(fit.array,1,odds.ratio)    # CM (given level of A)
     Yes       No
17.25144 17.25144   # these should be the same according to the model

apply(fit.array,2, odds.ratio)     # AM
     Yes       No
19.80646 19.80646

apply(fit.array,3, odds.ratio)     # AC
    Yes      No
7.80295 7.80295
```

To determine which dimension to apply the function `odds.ratio` over, I strongly recommend using R's version of `fitted`, which includes helpful variable labels instead of just levels (which are all the same, here).

To get marginal odds ratios, we just need to sum the array over the dimension we are excluding in the odds ratio. Here is a function that will sum the columns across an array.

```
sum.array<-function(array, perm=c(3,2,1)){
    res<-aperm(array,perm)
    colSums(res)
}
```

The `perm` argument is used to put the summed-over dimension in the rows place (place #1). In this way, when we do column sums, we sum over the rows. Had I used `rowSums` in the function, we would need to put the summed-over dimension in the columns place (place #2). The default is `c(3,2,1)`, which causes the matrices in the array to be summed, resulting in a single matrix which represents the sum. For example,

```
junk<-array(c(matrix(1:4,2,2)), dim=c(2,2,2))

, , 1

     [,1] [,2]
```

```
[1,]    1    3
[2,]    2    4

, , 2

     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

```
sum.array(junk)
```

```
     [,1] [,2]
[1,]    2    4
[2,]    6    8
```

Now, we compute the marginal odds ratios. The current dimensions of `fit.array` are A,C,M.

```
odds.ratio(sum.array(fit.array))  # AC (sum over M, so 3 needs to be listed first)
[1] 17.70244

odds.ratio(sum.array(fit.array, perm=c(1,2,3)))  # CM (sum over A, so 1 is first)
[1] 25.13620

odds.ratio(sum.array(fit.array, perm=c(2,1,3))) # AM (sum over C, so 2 is first)
[1] 61.87182
```

As `loglm` is a front-end to `loglin`, there is no need to show the results from `loglin`. `loglin` requires input of a table in the form of an array and also has an argument `margin`, where the model is specified using a list of numeric vectors. As an example, to fit the homogeneous association model, we do

```
loglin(fitted(fitACM), margin=list(c(1,2), c(2,3), c(1,3)), param=T,fit=T)

$lrt:
[1] 0.3738868

$pearson:
[1] 0.4011037

$df:
[1] 1

$margin:
$margin[[1]]:
[1] "alcohol"   "cigarette"

$margin[[2]]:
[1] "cigarette" "marijuana"

$margin[[3]]:
[1] "alcohol"   "marijuana"


$fit:

, , Yes
           Yes        No
Yes 910.382996 44.617058
 No   3.616851  1.383149

, , No
          Yes       No
Yes 538.61639 455.3836
 No  42.38408 279.6159
```

The `margin` argument gives the associations allowed for the model. Here, we choose all pairwise associations. `lrt` and `pearson` give the associated goodness-of-fit statistics, and `fit` gives the fitted counts.

The same model can be fit using `glm` with a `poisson` family. Here, the fit algorithm is Newton-Raphson.

```
options(contrasts=c("contr.treatment","contr.poly"))     # dummy coding for factors
(fit.glm<-glm(count~.^2, data=table.8.3, family=poisson))

Coefficients:
 (Intercept) marijuana cigarette  alcohol marijuana:cigarette marijuana:alcohol
     5.63342 -5.309042 -1.886669 0.487719            2.847889          2.986014

 cigarette:alcohol
          2.054534

Degrees of Freedom: 8 Total; 1 Residual
Residual Deviance: 0.3739859
```

The residual deviance is the likelihood ratio statistic. The sum of the squared pearson residuals gives the Pearson chi-squared statistic.

```
sum(resid(fit, type="pearson")^2)
[1] 0.4011004
```

C. Inference for Loglinear Models

Likelihood ratio chi-squared test statistics are output using the `summary` methods for `loglm` and `glm` and the `print` method for `loglin`. For example, the `summary` method for `loglm` gives the statistics and p-values as compared to a chi-squared distribution with the appropriate degrees of freedom.

```
summary(fitAC.AM.CM)    # homogeneous association model

Formula:
count ~ alcohol + cigarette + marijuana + alcohol:cigarette + alcohol:marijuana +
    cigarette:marijuana

Statistics:
                      X^2 df  P(> X^2)
Likelihood Ratio 0.3742223  1 0.5407117
         Pearson 0.4011002  1 0.5265216
```

Comparison of nested models can be done using the `anova` method. For example,

```
anova(fitAC.M, fitAC.AM.CM, fitAM.CM, fitA.C.M)

LR tests for hierarchical log-linear models

Model 1:
 count  ~  alcohol + cigarette + marijuana
Model 2:
 count  ~  alcohol + cigarette + marijuana + alcohol:cigarette
Model 3:
 count  ~  alcohol + cigarette + marijuana + alcohol:marijuana + cigarette:marijuana
Model 4:
 count  ~  alcohol + cigarette + marijuana + alcohol:cigarette + alcohol:marijuana +
    cigarette:marijuana

             Deviance df  Delta(Dev) Delta(df) P(> Delta(Dev)
```

```
  Model 1 1286.0200195  4
  Model 2  843.8267822  3 442.1932373         1        0.00000
  Model 3  187.7544556  2 656.0723267         1        0.00000
  Model 4    0.3742223  1 187.3802338         1        0.00000
Saturated    0.0000000  0   0.3742223         1        0.54071
```

gives the likelihood ratio tests comparing hierarchical loglinear models given in the list of arguments. Each item in the `Delta(Dev)` column compares `Deviances` between the current row and the previous row. So, the test of conditional independence between A and C, which compares models AM.CM and AM.CM.AC is 187.75 – 0.37 = 187.38. According to the output, the only model that fits well among the four is the homogeneous association model (Model 4). It's deviance is close enough to the deviance for the saturated model (zero) to give a nonsignificant p-value.

The parameter estimates returned by `loglin` and `loglm` (`fit$param`) come from a parameterization such that the `constant` component describes the overall mean, each single factor sums to zero, each two factor parameter sums to zero both by rows and columns, etc. Thus, the parameterization is not such that the parameters in the last row and the last column are zero, and will not match those from SAS on p. 325 in Agresti. To obtain parameter estimates and standard errors from `loglm` matching those from SAS, we can use plug-in calculations, as mentioned by Agresti. That is, take the fitted expected counts from `loglm` output and plug them into the formula (8.25) on p. 339 in Agresti. (This is illustrated below)

However, we can get the parameter estimates and standard errors directly from the `glm` fit. In order that our estimates match the SAS results given by Agresti on p. 325, we must first issue the command `options(contrasts= c("contr.treatment", "contr.poly"))`, and then ensure that our factor levels match those of SAS. An inspection of the `model.matrix` from `fit.glm` shows the dummy coding used by `contr.treatment`

```
model.matrix(fit.glm)
```

```
  (Intercept) marijuana cigarette alcohol marijuana:cigarette marijuana:alcohol cigarette:alcohol
1           1         0         0       0                    0                 0                 0
2           1         1         0       0                    0                 0                 0
3           1         0         1       0                    0                 0                 0
4           1         1         1       0                    1                 0                 0
5           1         0         0       1                    0                 0                 0
6           1         1         0       1                    0                 1                 0
7           1         0         1       1                    0                 0                 1
8           1         1         1       1                    1                 1                 1
```

By the manner in which `table.8.3` was set up, we see that the coding is 0 = Yes and 1 = No. We want 1 = Yes and 0 = No. Fortunately, we don't have to redefine the factors in `table.8.3`. We just need to use the `contrasts` argument of `glm`, as follows.

```
fit.glm2 <- update(fit.glm, contrasts = list(alcohol = as.matrix(c(1, 0)), marijuana =
   as.matrix(c(1, 0)), cigarette = as.matrix(c(1, 0))))
```

In the above, I reset the dummy coding to be 1 = Yes, 0 = No. In general, `contrasts` argument is a list with each element matching the name of a variable in the model. The value of an element in the list is a matrix of contrasts with number of rows equal to the number of levels of the variable.

Now, we can get estimates and ASEs to match those of SAS.

```
summary(fit.glm2, cor = F)
```

```
Coefficients:
                      Value Std. Error    t value
        (Intercept)  5.633420 0.05970077  94.360930
          marijuana -5.309042 0.47506865 -11.175316
          cigarette -1.886669 0.16269584 -11.596294
            alcohol  0.487719 0.07576708   6.437083
marijuana:cigarette  2.847889 0.16383796  17.382353
```

```
  marijuana:alcohol  2.986014 0.46454749   6.427791
  cigarette:alcohol  2.054534 0.17406289  11.803401

(Dispersion Parameter for Poisson family taken to be 1 )

    Null Deviance: 2851.461 on 7 degrees of freedom

Residual Deviance: 0.3739859 on 1 degrees of freedom

Number of Fisher Scoring Iterations: 3
```

For `loglm`, we compute equation (8.25) using the `model.matrix` function to get **X**. Note that we still need to modify the 0/1 coding using `contrasts` argument to match the coding in Agresti.

```
options(contrasts=c("contr.treatment","contr.poly"))     # ensure we have treatment
   contrasts

X<-model.matrix(count~(alcohol+cigarette+marijuana)^2,data=table.8.3,
   contrasts=list(alcohol=as.matrix(c(1,0)), marijuana=as.matrix(c(1,0)),
   cigarette=as.matrix(c(1,0))))

sqrt(diag(solve(t(X)%*%diag(c(fitAC.AM.CM$fitted))%*%X)))
# R output
          (Intercept)              alcohol1            cigarette1
           0.05970110            0.47519394            0.16269591

           marijuana1   alcohol1:cigarette1   alcohol1:marijuana1
           0.07576733            0.16383935            0.46467452

cigarette1:marijuana1
           0.17406330
```

As a side note, we in fact do not have to use the `contrasts` argument to `model.matrix` or to `glm` if we specify the levels of the variables in `table.8.3` at the outset using the `factor` function. For example, we did

```
table.8.3<-data.frame(expand.grid(marijuana=factor(c("Yes","No"), levels=c("No",
   "Yes")), cigarette=factor(c("Yes","No"),levels=c("No", "Yes")),
   alcohol=factor(c("Yes","No"),levels=c("No","Yes"))),
   count=c(911,538,44,456,3,43,2,279))
```

which gives the coding we want: 1=Yes, 0=No. However, it is instructive to see the `contrasts` argument in its own right.

D. Loglinear Models for Higher Dimensions

Fitting of loglinear models with more than three dimensions is straightforward. Agresti uses a four-way table displaying counts of accidents that either did or did not involve injury, and whether the driver wore a seat belt, their gender, and location (rural, urban). We can fit several loglinear models specifying different degrees of conditional independence using `loglm`.

```
table.8.8<-data.frame(expand.grid(belt=c("No","Yes"), location=c("Urban","Rural"),
   gender=c("Female","Male"), injury=c("No","Yes")),
   count=c(7287,11587,3246,6134,10381,10969,6123, 6693,996, 759, 973, 757, 812, 380,
   1084, 513))
```

We fit mutual independence model, a model with all pairwise interactions and no higher association (i.e., each pair of variables has the same odds ratio at each combination of the other two variables), and the homogeneous three-way association model (i.e., no four-way interaction).

```
library(MASS)
```

```
fitG.I.L.S<-loglm(count~., data=table.8.8, fit=T, param=T)  # mutual independence
fitGI.GL.GS.IL.IS.LS<-update(fitG.I.L.S, .~.^2, data=table.8.8, fit=T, param=T) # all
   pairwise associations
fitGIL.GIS.GLS.ILS<-update(fitG.I.L.S, .~.^3, data=table.8.8, fit=T, param=T)   # all
   three-way associations
```

A comparison of likelihood ratio statistics (next) tells us that the three-way association model fits best, and is the only one to have a nonsignificant p-value.

```
anova(fitG.I.L.S, fitGI.GL.GS.IL.IS.LS, fitGIL.GIS.GLS.ILS)

LR tests for hierarchical log-linear models

Model 1:
 count  ~  belt + location + gender + injury
Model 2:
 count  ~  belt + location + gender + injury + belt:location + belt:gender +
   belt:injury + location:gender + location:injury + gender:injury
Model 3:
 count  ~  belt + location + gender + injury + belt:location + belt:gender +
   belt:injury + location:gender + location:injury + gender:injury +
   belt:location:gender + belt:location:injury + belt:gender:injury +
   location:gender:injury

           Deviance df Delta(Dev) Delta(df) P(> Delta(Dev)
  Model 1 2792.76245 11
  Model 2   23.35137  5 2769.41113         6        0.00000
  Model 3    1.32489  1   22.02648         4        0.00020
Saturated    0.00000  0    1.32489         1        0.24972
```

However, we would like a simpler model than all three-way interactions. Here is a fit of the all-pairwise model plus the three-way interaction GLS.

```
(fitGI.IL.IS.GLS <- update(fitGI.GL.GS.IL.IS.LS, . ~ . + gender:location:belt))

Statistics:
                      X^2 df  P(> X^2)
Likelihood Ratio 7.462791  4 0.1133613
         Pearson 7.487374  4 0.1122673
```

The p-value is not significant, and this model is simpler to interpret than all three-way interactions. This model says that whether or not injury occurred, the association between any two of the remaining variables changes at each level of the third variable. For example, the association between gender and seat belt use is not the same across urban and rural locations. However, any conditional odds ratio between injury and another variable *is* the same at the combinations of the other two variables (because injury does not appear in the three-way interaction). So, for example, GI is the same no matter what the levels of L and S.

Here are the fitted counts from this model.

```
fitted(fitGI.IL.IS.GLS) # output is from R

Re-fitting to get fitted values

, , gender = Female, injury = No

     location
belt      Urban    Rural
  No   7273.214 3254.662
  Yes 11632.622 6093.502

, , gender = Male, injury = No
```

```
     location
belt     Urban    Rural
  No  10358.93 6150.193
  Yes 10959.23 6697.643

, , gender = Female, injury = Yes

     location
belt      Urban    Rural
  No  1009.7857 964.3376
  Yes  713.3783 797.4979

, , gender = Male, injury = Yes

     location
belt     Urban     Rural
  No  834.0684 1056.8071
  Yes 389.7681  508.3566
```

We can examine the three-way interaction by seeing how the conditional odds ratios of any two variables of the trio (gender, location, belt use) change at the levels of the third variable. But, they are the same regardless of injury status. (Output is given from R)

```
fit.array<-fitted(fitGI.IL.IS.GLS)
odds.ratio<-function(x) x[1,1]*x[2,2]/(x[2,1]*x[1,2])

apply(fit.array,c(1,4),odds.ratio)    # GL S (same for I = yes or no - column)

     injury
belt        No      Yes
  No  1.326766 1.326766
  Yes 1.166682 1.166682

apply(fit.array,c(2,4),odds.ratio)    # GS L (same for I = yes or no - column)

        injury
location        No       Yes
   Urban 0.6614758 0.6614758
   Rural 0.5816641 0.5816641

apply(fit.array,c(3,4),odds.ratio)    # LS G (same for I = yes or no - column)

        injury
gender         No      Yes
  Female 1.170603 1.170603
  Male   1.029362 1.029362
```

For interpretation, we take the GS L = urban odds ratio, which is 0.66. This is the estimated odds that males used seat belts over females when accidents occurred in urban locations. Thus, females are about 1.0/0.66 = 1.5 times more likely to have used a seat belt when an accident occurred in an urban area, regardless of whether there was an injury. The analogous odds ratio at rural locations is quite similar (0.58), however. As this similarity between Urban and Rural carries over to other odds ratio comparisons (i.e., the GL S and LS G comparisons), the three-way interaction may not be necessary in this loglinear model.

We can also see that the conditional odds ratio between injury and any other variable are the same at the combinations of the remaining two.

```
apply(fit.array,c(1,2),odds.ratio)    # GI (same for each combination of LS)

        Urban     Rural
 No 0.5799410 0.5799411
```

```
Yes 0.5799411 0.5799412

apply(fit.array,c(1,3),odds.ratio)    # IL (same for each combination of GS)

      Female     Male
 No 2.134127 2.134127
Yes 2.134127 2.134127

apply(fit.array,c(2,3),odds.ratio)    # IS (same for each combination of GL)

         Female      Male
Urban 0.4417123 0.4417123
Rural 0.4417122 0.4417123
```

Computing the dissimilarity matrix to check goodness-of-fit of this model to the data is simple in S.

```
Fitted.values <- c(fit.array)
sum(abs(table.8.8$count - Fitted.values))/(2 * sum(table.8.8$count))

[1] 0.002507361
```

Approximate standard errors for these models can be obtained using the "brute force" calculations mentioned above (see pp. 338-339 of Agresti and pp. 28-29 of Thompson, 1999) or by using `glm` with poisson family to fit the model (see previous section B, above).

E. Loglinear-Logit Model Connection

The correspondence between a loglinear model with three variables and a logit model with one of those variables a response is shown in equation (8.15) of Agresti. The four-way loglinear model that was fit to the auto accident data (GI, LI, IS, GLS) is equivalent to the logit model (G + L + S) with response I. This is because any term in the logit model that does not have the symbol I disappears. Thus, GLS disappears, leaving the symbols (GI, LI, IS), which together specify that given injury, G, L, and S are independent. The equivalent logit model is then that G, L, and S are independent in their effects on I. To get the connections between the parameter estimates, we can fit both models using `glm`. We use treatment contrasts.

```
options(contrasts = c("contr.treatment", "contr.poly"))

fit.loglinear <- glm(count ~ .^2 + gender:location:belt, data = table.8.8, family =
   poisson)

fit.logit <- glm(injury ~ gender + belt + location, data = table.8.8, family =
   binomial, weight = count)

fit.loglinear$coefficients

 (Intercept)      belt   location    gender   injury belt:location belt:gender belt:injury
    8.891954 0.4696151 -0.8041099 0.3536508 -1.97446     0.1575195   -0.413282  -0.8170974

location:gender location:injury
      0.2827442       0.7580583

 gender:injury gender:location:belt
    -0.5448292           -0.1285802

fit.logit$coefficients

 (Intercept)    gender       belt location
    -1.97446 -0.544829 -0.8170971 0.758058
```

Note that for the logit model, all terms in the loglinear fit that did not have injury (I) are removed. The remaining coefficients become the coefficients for the logit model. The exact correspondence among

these remaining coefficients is a consequence of having only two levels per variable and also of using treatment contrasts. To see that this is so, read subsection 8.5.2 in Agresti, and recall that we set the last coefficient in each row and column of a pairwise interaction to zero for constraints.

The fact that there is a correspondence between estimates from the two models, which assume two different sampling distributions for responses, follows from the correspondence of the likelihoods with respect to the parameters in question.

F. Contingency Table Standardization

Table standardization is useful for comparing tables with different marginal totals or matching sample table data to standardized marginal distributions. Next, we show how to "rake" Table 8.15 by forcing all row and column totals to equal 100. This is done using the tip on p. 346 of Agresti. We make the log of the observed count an offset in a glm and create a pseudo response that satisfies the model and the marginal totals.

First I input the data, coding factors with levels in the order most natural.

```
table.8.15<-data.frame(expand.grid(Attitude=factor(c("Disapprove","Middle","Approve"),
   levels=c("Disapprove","Middle","Approve")),
Education=factor(c("<HS","HS",">HS"), levels=c("<HS","HS",">HS"))),
count=c(209,101,237,151,126,426,16,21,138))
```

Then, we fit an independence model using pseudo-values 100/3 for each cell.

```
fit <- glm(I(rep(100/3, 9)) ~ Attitude + Education + offset(log(count)), family =
   poisson, data = table.8.15)
cbind(table.8.15, std = fitted(fit))
```

```
    Attitude Education count      std
1 Disapprove       <HS   209 49.42773
2     Middle       <HS   101 32.01875
3    Approve       <HS   237 18.55352
4 Disapprove        HS   151 32.76098
5     Middle        HS   126 36.64454
6    Approve        HS   426 30.59448
7 Disapprove       >HS    16 17.81129
8     Middle       >HS    21 31.33671
9    Approve       >HS   138 50.85200
```

In R, using 100/3 as a count gives a warning because it is not an integer. However, the estimation continues, apparently still using 100/3.

Chapter 9 –Building and Extending Loglinear Models

A. Summary of Chapter 9, Agresti

This chapter extends loglinear models to ordinal variables and correlated variables, and also deals with special issues like collapsibility and empty cells.

With higher order tables, the analysis is easier if we can collapse over some dimensions (variables). Collapsibility describes the conditions under which a variable can be collapse over (i.e., ignored). For three-way tables with variables X, Y, and Z, we can collapse over Z while keeping the XY association the same for the marginal table as well as the conditional table (on levels of Z) if either Z and X are conditionally independent given Y or Z and Y are conditionally independent given X. For four-way tables with variables W, X, Y, and Z, if we collapse over Y, and association terms involving XY and WY are zero, then the partial WX odds ratios, conditioning on Y, are the same as the marginal WX odds ratios after collapsing. The general principle describing these examples is on p. 360 of Agresti.

Loglinear models treat all variables as responses. However, sometimes certain marginal totals are fixed by sampling design. If the corresponding loglinear terms are not included in the model, then the fitted marginal totals will not necessarily match the observed totals for these combinations. This is because the likelihood will not match the observed total to the fitted total in the likelihood equation (equation 8.22 in Agresti). In sum, the highest interaction term containing all explanatory variables should be included in a loglinear model.

Nested loglinear models can be compared using the likelihood ratio statistic in equation (9.3) in Agresti, or using the Pearson statistic in equation (9.4). These have asymptotic chi-squared distributions, and are used to test the significance of an additional term or variable in the model. To check the fit of a given model, one can examine cell residuals.

Just as with logit models, if there are ordinal responses, then greater power results from fitting specific ordinal trends than either ignoring ordinality or fitting general association terms. The latter situation may not even be possible without fitting a saturated model. The *linear-by-linear association* (*L* x *L*) model assigns ordered row scores to rows and ordered column scores to columns, and includes, in addition to main effect terms for the rows and columns, an interaction term in the row and column scores. The interaction term has a single unknown parameter which represents positive or negative linear association, depending on the sign of the parameter. Odds ratios are a function of the distance between corresponding rows and columns and the magnitude of the interaction parameter.

L x *L* models are best fit when the observed counts show a linear trend in the rows at each column and a linear trend in the columns for each row. The model is fit using ML estimation, and the correlation between rows and columns is the same in both the observed and fitted counts.

Association models generalize loglinear models for ordinal responses to multi-way tables and to mixed ordinal and nominal responses. The row effects model and column effects model have either rows or columns as nominal responses, respectively, and the other as ordinal. The nominal variable has parameter scores instead of fixed scores. Ordinal responses within multi-way tables are straightforward generalizations of their two-way table analogs.

Other types of models that emphasize the estimation of association are Multiplicative Row and Column effects models (RC Models), correlation models, and correspondence analysis. The RC model modifies the *L* x *L* model by replacing the scores with parameters. Correlation models are similar to RC models. In these models, each cell probability is augmented with an association term, dependent on row and column scores. Correlation models can have either fixed or parameter scores. Correspondence analysis is mostly a graphical technique to represent associations in contingency tables.

The Poisson loglinear model can also be used to model the rate of occurrence of an event over an exposure time or space, dependent on covariates. The exposure is treated as an offset. Also, due to the connection between ML estimation using a Poisson likelihood for numbers of events and a negative exponential likelihood for survival time, one can model survival times using a Poisson loglinear model.

Sparse tables have many empty cells where there are no observations. A *sampling zero* occurs in a cell when the data do not contain observations corresponding to that cell. A *structural zero* occurs when it is impossible for the data to have an observation for the cell. For unsaturated models, MLEs exist when all cell counts are positive, but do not exist when some counts in the sufficient marginal tables are

zero. They also may not exist when all sufficient counts are positive, but at least one cell is zero. However, even if a point estimate is infinite, one endpoint of a likelihood-ratio confidence interval is usually finite. Adding a constant like 0.5 to each cell to prevent sampling zeroes for fitting an unsaturated model is not recommended, especially for large numbers of cells, because it influences estimates and test statistics conservatively.

As an alternative to large-sample goodness-of-fit tests (which may not apply when a table is sparse), one can use exact tests or Monte Carlo approximations to exact tests.

B. Model Selection and Comparison

Agresti uses the Student Survey example to illustrate model selection and comparison. We will use a backwards stepwise procedure to help in comparing models by comparing likelihood ratio chi-square statistics.

We start by inputing the data

```
table.9.1<-data.frame(expand.grid(cigarette=c("Yes","No"),
    alcohol=c("Yes","No"),marijuana=c("Yes","No"), sex=c("female","male"),
race=c("white","other")),count=c(405,13,1,1,268,218,17,117,453,28,1,1,228,201,17,133,2
    3,2,0, 0,23, 19,1,12,30,1,1,0,19,18,8,17))
```

Now, we fit some loglinear models using `glm` and the `poisson` family. Note that we must include the highest interaction involving explanatory variables. Thus, `sex:race` is always included. The next two models fit mutual independence + GR (Agresti uses G for gender and R for race) and homogeneous association.

```
fit.GR<-glm(count~ . + sex*race,data=table.9.1,family=poisson)  # mutual independence +
    GR
fit.homog.assoc<-glm(count~ .^2,data=table.9.1,family=poisson)  # homogeneous
    association
```

To reproduce Table 9.2 in Agresti, we can start by eliminating each two-way interaction in turn (with exception of sex:race) and comparing likelihood ratio statistics with and without these terms. I use `stepAIC`, which is part of library `MASS` in both R and S-PLUS. This will remove terms based on AIC, which is a function of both the residual deviance (likelihood-ratio chi-square) and number of terms in the model. Thus, it does not quite select the "best" model that Agresti's Table 9.2 does.

```
library(MASS)

summary(res<-stepAIC(fit.homog.assoc, scope= list(lower = ~ + cigarette + alcohol +
    marijuana + sex*race), direction="backward"))  # note that I have saved the final
    model in an object called "res"


Start:  AIC= 47.34
 count ~ (cigarette + alcohol + marijuana + sex + race)^2

                    Df  Deviance       AIC
   - cigarette:race  1  15.78347  45.78347
    - cigarette:sex  1  16.31718  46.31718
               <none> NA  15.34034  47.34034
      - alcohol:sex  1  18.71695  48.71695
   - marijuana:race  1  18.92894  48.92894
     - alcohol:race  1  20.32086  50.32086
    - marijuana:sex  1  25.16101  55.16101
- alcohol:marijuana  1 106.95800 136.95800
- cigarette:alcohol  1 201.19931 231.19931
- cigarette:marijuana  1 513.47218 543.47218
```

```
Step:  AIC= 45.78
 count ~ cigarette + alcohol + marijuana + sex + race + cigarette:alcohol +
    cigarette:marijuana + cigarette:sex + alcohol:marijuana +
    alcohol:sex + alcohol:race + marijuana:sex + marijuana:race + sex:race

                      Df  Deviance       AIC
      - cigarette:sex  1  16.73504  44.73504
                <none> NA  15.78347  45.78347
     - marijuana:race  1  18.95695  46.95695
        - alcohol:sex  1  19.18046  47.18046
       - alcohol:race  1  20.33869  48.33869
      - marijuana:sex  1  25.56924  53.56924
  - alcohol:marijuana  1 107.79480 135.79480
  - cigarette:alcohol  1 201.21688 229.21688
- cigarette:marijuana  1 513.50057 541.50057

Step:  AIC= 44.74
 count ~ cigarette + alcohol + marijuana + sex + race + cigarette:alcohol +
    cigarette:marijuana + alcohol:marijuana + alcohol:sex +
    alcohol:race + marijuana:sex + marijuana:race + sex:race

                      Df  Deviance       AIC
              <none> NA  16.73504  44.73504
     - marijuana:race  1  19.90859  45.90859
       - alcohol:race  1  21.29033  47.29033
        - alcohol:sex  1  22.02148  48.02148
      - marijuana:sex  1  25.81125  51.81125
  - alcohol:marijuana  1 108.77579 134.77579
  - cigarette:alcohol  1 204.11536 230.11536
- cigarette:marijuana  1 513.73033 539.73033

Call: glm(formula = count ~ cigarette + alcohol + marijuana + sex + race +
    cigarette:alcohol + cigarette:marijuana + alcohol:marijuana +
    alcohol:sex + alcohol:race + marijuana:sex + marijuana:race + sex:race, family =
    poisson, data = table.9.1)

Coefficients:
                       Value Std. Error    t value
       (Intercept)  2.2651798 0.12467553  18.168600
         cigarette -0.2822714 0.05491143  -5.140485
           alcohol -1.4163287 0.12121797 -11.684149
         marijuana  1.2671910 0.12443051  10.183925
               sex  0.1090531 0.04581267   2.380413
              race -1.1940934 0.05467500 -21.839842
  cigarette:alcohol  0.5136335 0.04351564  11.803423
cigarette:marijuana  0.7119723 0.04095963  17.382295
  alcohol:marijuana  0.7486558 0.11619789   6.442938

                     Value Std. Error    t value
   alcohol:sex  0.07321130 0.03190131  2.2949310
  alcohol:race  0.11284083 0.05166169  2.1842266
 marijuana:sex -0.06805431 0.02261757 -3.0089132
marijuana:race  0.07958278 0.04495553  1.7702554
      sex:race  0.03582621 0.03976789  0.9008829

(Dispersion Parameter for Poisson family taken to be 1 )

    Null Deviance: 4818.051 on 31 degrees of freedom

Residual Deviance: 16.73504 on 18 degrees of freedom
```

The output is lengthy, but it indicates the eliminated coefficient (which I've bolded) at each step. Note that we don't quite get to Agresti's Model 6, but stop at Model 5. Model 6 would correspond to the removal:

```
                      Df  Deviance       AIC
```

```
     - marijuana:race  1  19.90859  45.90859
```

We now rename the result of the stepwise search:

```
fit.AC.AM.CM.AG.AR.GM.GR<-res     # Agresti's Model 5
```

The `update` method for `glm` can be used to remove additional specified terms. For example, to fit Agresti's Model 6,

```
fit.AC.AM.CM.AG.AR.GM.GR<-update(fit.AC.AM.CM.AG.AR.GM.GR.MR, ~. - marijuana:race)
```

According the conditional associations that Model 6 implies, cigarette use is independent of both gender and race, given alcohol and marijuana use. Thus, according to this model, we don't need to consider the association between cigarette use and both gender and race if alcohol and marijuana use are already considered. Furthermore, if we collapse over gender and race, the conditional associations between cigarette use and alcohol and between cigarette use and marijuana are the same as in the homogeneous association model (AC, AM, CM) that was fit in Chapter 8 (i.e., **`fitAC.AM.CM`** in section C).

C. Diagnostics for Checking Models

Pearson residuals for loglinear models use an estimate of the Poisson standard error in the denominator. Thus, they differ from Pearson residuals for logit models, which use an estimate of a binomial standard error in the denominator. As in Chapter 4, standardized versions of the residuals multiply the denominator by a function of the leverages from the hat matrix. This results in the variance of the residual being 1.0, and an asymptotic normal distribution. Thus, residuals greater than about 3 in magnitude are considered large.

To get standardized Pearson residuals for Model 6 from the Student Survey Example above, we use the `resid` function. The only large residual is highlighted.

```
res.model6<-resid(fit.AC.AM.CM.AG.AR.GM.GR, type="pearson")/sqrt(1-
   lm.influence(fit.AC.AM.CM.AG.AR.GM.GR.MR)$hat)
fit.model6<-fitted(fit.AC.AM.CM.AG.AR.GM.GR)

data.frame(table.9.1, fitted=fit.model6, residuals=res.model6)
```

```
  cigarette alcohol marijuana    sex  race count       fitted   residuals
1       Yes     Yes       Yes female white   405 394.81378138  2.31218352
2        No     Yes       Yes female white    13  19.34936858 -1.95386309
3       Yes      No       Yes female white     1   1.25568751 -0.26793681
4        No      No       Yes female white     1   0.48020744  0.79799852
5       Yes     Yes        No female white   268 267.43245628  0.07000670
6        No     Yes        No female white   218 226.10552205 -1.00689966
7       Yes      No        No female white    17  17.18531924 -0.06124557
8        No      No        No female white   117 113.37765755  0.94414973
9       Yes     Yes       Yes   male white   453 452.25694845  0.16765462
10       No     Yes       Yes   male white    28  22.16459202  1.77790180
11      Yes      No       Yes   male white     1   1.92669440 -0.87173015
12       No      No       Yes   male white     1   0.73681787  0.33769336
13      Yes     Yes        No   male white   228 234.02052238 -0.74906553
14       No     Yes        No   male white   201 197.85680885  0.39389108
15      Yes      No        No   male white    17  20.14353803 -1.01794011
16       No      No        No   male white   133 132.89407806  0.02722599
17      Yes     Yes       Yes female other    23  27.53248411 -1.38158402
18       No     Yes       Yes female other     2   1.34933533  0.57577121
19      Yes      No       Yes female other     0   0.15851532 -0.40463832
20       No      No       Yes female other     0   0.06062037 -0.24784091
21      Yes     Yes        No female other    23  18.64950060  1.32549550
22       No     Yes        No female other    19  15.76755166  1.02007047
23      Yes      No        No female other     1   2.16943817 -0.84701196
```

```
24        No      No        No female other    12  14.31255445 -0.88482760
25       Yes     Yes       Yes   male other    30  35.77995571 -1.74121619
26        No     Yes       Yes   male other     1   1.75353441 -0.58857418
27       Yes      No       Yes   male other     1   0.27593320  1.41736166
28        No      No       Yes   male other     0   0.10552401 -0.32855531
29       Yes     Yes        No   male other    19  18.51435109  0.14864501
30        No     Yes        No   male other    18  15.65328710  0.74381812
31       Yes      No        No   male other     8   2.88487421  3.26629667
32        No      No        No   male other    17  19.03254028 -0.74988607
```

D. Modeling Ordinal Associations

When both responses in a loglinear model are ordinal, a linear-by-linear association model can be used to fit a linear trend in ordered row and column scores. These scores may be assumed to represent an underlying continuous distribution. For an *I* x *J* table, the loglinear model adds an interaction term in the scores, as follows

$$\log \mu_{ij} = \lambda + \lambda_i^X + \lambda_j^Y + \beta u_i v_j , \qquad i = 1,...,I;\ j = 1,...,J$$

The parameter, β, describes the linear association. It is positive or negative with the sign of β. Odds ratios depend on β, as well as the score distance between the corresponding rows and columns. When $\{u_i = i\}$ and $\{v_j = j\}$, the local odds ratios for adjacent rows and columns are uniform, and equal $\exp(\beta)$. This is called the *uniform association model*.

Agresti fits a uniform association (UA) model to data from the 1991 General Social Survey on opinions about premarital sex and birth control for teenagers. The four categories for opinions about premarital sex range from "Always wrong" to "Not wrong at all". The four categories for opinions about teenage birth control sex range from "Strongly disagree" to "Strongly agree". The scores used are {1, 2, 3, 4} for both rows and columns, initially.

The data are available from Agresti's web site. I have copied the data into a text file called `sex.txt`. We first read in the data, and give the columns names.

```
table.9.3<-read.table("c:/program files/r/rw1070/cda/sex.txt",
   col.names=c("premar","birth","u","v","count"))
```

Then, I noticed that "birth" is recorded so that the 1,1 cell corresponds to the upper right hand corner instead of upper left (as might seem more natural), so I perform a transformation of the birth column.

```
table.9.3$birth<-5-table.9.3$birth   # rearrange scores so that table starts at 1,1 in
   the upper left corner
```

I also noticed that the u and v scores in the data file are not uniform scores, but scores used later in the section. So, I next create uniform scores: u1 and v1.

```
table.9.3$u1<-table.9.3$premar
table.9.3$v1<-table.9.3$birth
```

Also, premar and birth need to be factors for the model fitting. I have coded the levels of the factors so that the 4th level is set to zero, as in the SAS analysis in Table 9.4 in Agresti.

```
table.9.3$premar<-factor(table.9.3$premar, levels=4:1)
table.9.3$birth<-factor(table.9.3$birth, levels=4:1)
```

Now, we fit the uniform association model. Note the use of the ":" term for the interaction between u1 and v1, which not factors.

```
options(contrasts=c("contr.treatment","contr.poly"))
(fit.ua<- glm(count ~ premar + birth + u1:v1, data=table.9.3, family=poisson))

Coefficients:
 (Intercept)     premar3   premar2  premar1   birth3   birth2   birth1     u1:v1
   0.4734922 -0.01633777 0.1077209 1.753685 1.155142 1.415559 1.879664 0.2858355

Degrees of Freedom: 16 Total; 8 Residual
Residual Deviance: 11.53369
```

The estimated local odds ratio for the UA model is

```
exp(coef(fit.ua)["u1:v1"])

    u1:v1
 1.330873
```

and a Wald 95% CI is obtained from the ASEs from the `summary` output.

```
exp(summary(fit.ua)$coef["u1:v1", 1] +
   1.96*c(-1,1)*summary(fit.ua)$coef["u1:v1", 2])
[1] 1.259221 1.406603
```

According to the estimate, the association is positive, implying that subjects with more favorable attitudes about teenage birth control also tended to have more tolerant attitudes about premarital sex. But, the assocation is rather weak. The odds of responding in the adjacent higher category on one variable are 1.33 times more likely if the subject responded in the higher category of the other variable (versus the lower category).

However, with distance in categories, the odds ratios are larger. For example, the odds of responding "Strongly agree" to teen birth control over "Strongly disagree" are 13.1 times higher if the subject responded that premarital sex was "Not wrong at all" versus responding that it was "Always wrong".

```
exp(coef(fit.ua)["u1:v1"] * (4 - 1) * (4 - 1))

    u1:v1
 13.09878
```

Fitted counts for the UA model are obtained using `fitted`. Note that since we reversed the "birth" variable, we must also reverse the fitted values. Thus, I have labeled the columns of the matrix from "Strongly Agree" to "Strongly Disagree" instead of the reverse labeling.

```
matrix(fitted(fit.ua), byrow = T, ncol = 4, dimnames = list(PreMar = c("Always wrong",
   "Almost always wrong", "Wrong sometimes",
   "Not wrong"), "TeenBirth" = c("SA", "A", "D", "SD")))

                         SA         A        D       SD
      Always wrong  29.09363  69.39574 67.65406 80.85657
Almost always wrong  17.59996  31.54350 23.10650 20.75004
    Wrong sometimes  48.77315  65.68137 36.15178 24.39370
          Not wrong 155.53326 157.37940 65.08766 32.99969
```

Standardized Pearson residuals can be obtained using the code from Section C of this chapter.

The deviance from a UA model is much smaller than that from an independence model:

```
(fit.ind<-glm(count ~ premar + birth, data=table.9.3, family=poisson))
…
```

```
Degrees of Freedom: 16 Total; 9 Residual
Residual Deviance: 127.6529
```

A LR test of the null hypothesis of independence (i.e., $H_0 : \beta = 0$) has df =1:

```
anova(fit.ind, fit.ua, test = "Chi") # output from S-PLUS

Analysis of Deviance Table

Response: count

                    Terms Resid. Df Resid. Dev   Test Df Deviance Pr(Chi)
1         premar + birth         9   127.6529
2 premar + birth + u1:v1         8    11.5337 +u1:v1  1 116.1192       0
```

E. Association Models

Association models generalize loglinear models to ordinal responses, and include the linear-by-linear association models. The focus of these models is on estimating association. Row and column effects models have rows or columns, respectively, as nominal responses, and model an ordinal effect at each of the rows or columns. Multi-way tables with some ordinal responses generalize the two-way tables, adding three or higher way interactions.

1. Row and Column Effects Models

The row effect model treats rows (X) as nominal and columns (Y) as ordinal. Each row has a parameter associated with it, called the row effect. The model for the log expected count in cell *ij* is

$$\log \mu_{ij} = \lambda + \lambda_i^X + \lambda_j^Y + \mu_i v_j$$

where the μ_i are the unknown row effects (i = 1,…, I), and the v_j are known column scores. Thus, the model has $I - 1$ more parameters than an independence model. An independence model here would imply that all row effects are equal. That is, the linear effect of the column variable on the log expected count is the same within each row. The column effects model is defined similarly.

For the row effects model, the distance between adjacent-categories logits is the same across rows. However, the level of the logit will differ between rows i and k by $\mu_i - \mu_k$. Thus, this model is called the *parallel odds model*.

The Political Ideology Example (Table 9.5 in Agresti) uses a table that cross-classifies Party Affiliation (Democrat, independent, Republican) and Political Ideology (Liberal, Moderate, Conservative) for a sample of voters in a Wisconsin primary.

A row effects model is fit to Table 9.5, with rows as Party Affiliation. When we construct the table, the variable `c.Ideo` are the scores (1, 2, 3) for Political Ideology.

```
table.9.5<-
   data.frame(expand.grid(Affil=factor(c("Democrat","Independent","Republican"),
   levels=c("Republican","Independent","Democrat")),
    Ideology=factor(c("Liberal","Moderate","Conservative"),
   levels=c("Conservative","Moderate","Liberal"))),
   c.Ideo=rep(1:3,each=3), count=c(143,119,15,156,210,72,100,141,127))
```

I coded the levels of the factor variables so that the Republican and Conservative levels had zero-valued coefficients, for identifiability. This is done to match the SAS results in Table 9.6 in Agresti.

To fit a row effects model:

```
options(contrasts = c("contr.treatment", "contr.poly"))
(fit.RE <- glm(count ~ Ideology + Affil * c.Ideo, family = poisson, data = table.9.5))
```

```
Coefficients: (1 not defined because of singularities)
 (Intercept)   Moderate    Liberal     Indep      Demo       c.Ideo  Indepc.Ideo    Demc.Ideo
    4.856484  -0.6244464  -2.048811   2.953568   3.322995     NA     -0.9426178   -1.213361

Degrees of Freedom: 9 Total; 2 Residual
Residual Deviance: 2.814931
```

The `c.Ideo` main effect is confounded with `Ideology`. Thus, its coefficient is not estimable. Since the Republican and Conservative category coefficients are set to zero, the two row effect estimates indicate deviation from Conservativism relative to Republicans. The Republican row effect is thus at zero, and Democrats are 1.21 units in the Liberal direction. Independents are 0.94 units in the Liberal direction.

We can get predicted logits (equation 9.9 in Agresti) using `fit.RE$linear.predictors` and taking differences:

```
res<-matrix(fit$linear.predictors,byrow=F,ncol=3)
mat<-matrix(c(res[,2]-res[,1],res[,3]-res[,2]),byrow=T,ncol=3)
```

```
           [,1]       [,2]      [,3]
[1,]  0.2110031  0.4817465 1.4243643
[2,] -0.5889149 -0.3181714 0.6244464
```

Thus, the odds that Republicans are conservative instead of moderate (exp(1.424)), or moderate instead of liberal (exp(0.211)) are exp(1.424 – 0.211) = exp(1.213) = 3.36 times the corresponding estimated odds for Democrats, and exp(1.424 – 0.482) = exp(0.942) = 2.57 times the corresponding estimated odds for independents.

The fitted values

```
matrix(fitted(fit.RE), byrow = F, ncol = 3, dimnames = list(Affiliation =
   c("Democrat", "Independent", "Republican"), Ideology = c("Liberal", "Moderate",
   "Conservative")))
```

```
             Liberal  Moderate Conservative
   Democrat 136.63414 168.73171     93.63414
Independent 123.79454 200.41091    145.79454
 Republican  16.57131  68.85738    128.57131
```

To compare independence and row effects models, use `anova`:

```
fit.ind<-glm(count~Affil+Ideology,family=poisson,data=table.9.5)
anova(fit.ind,fit.RE)
```

```
Analysis of Deviance Table

Response: count

                      Terms Resid. Df Resid. Dev          Test Df Deviance
1          Affil + Ideology         4   105.6622
2 Ideology + Affil * c.Ideo         2     2.8149 +Affil:c.Ideo  2 102.8472
```

2. Ordinal Models in Multi-way Tables

An extension of association models to multi-way tables permits higher-order interaction models that are more parsimonious than their corresponding nominal models. In particular, the three-way interaction model for a three-way table is not saturated. For a three-way table, with X and Y ordinal, with scores $\{u_i\}$ and $\{v_j\}$ respectively, we model *heterogeneous linear-by-linear* XY association as

$$\log \mu_{ijk} = \lambda + \lambda_i^X + \lambda_j^Y + \lambda_k^Z + \beta_k u_i v_j + \lambda_{ik}^{XZ} + \lambda_{jk}^{YZ}$$

In this model, the log XY odds ratios are uniform within each level of Z, but differ across levels of Z. If $\beta_k = \beta$ for all *k*, then the log odds ratios the same across *k* as well. The model is then called the *homogeneous linear-by-linear association* model.

The data in Table 9.7 in Agresti are used to fit these models. The table cross-classifies smoking status (S), breathing test results (B) and age (A) for workers in industrial plants.

The data are read in, and the levels are set so that the last category is zero. I also define scores for all three variables. These are denoted with a "c" prefix.

```
table.9.7<-data.frame(expand.grid(Status=factor(c("Never","Former","Current"),
      levels=c("Current","Former","Never",)),
   Breath=factor(c("Normal","Borderline","Abnormal"),
      levels=c("Abnormal","Borderline","Normal")),
   Age=factor(c("< 40","40-50"), levels=c("40-50","< 40"))),
   c.Status=rep(1:3, 6), c.Breath=rep(rep(1:3,each=3),2), c.Age=rep(1:3,each=6),
   count=c(577,192,682,27,20,46,7,3,11,164,145,245,4,15,47,0,7,27)
   )
```

The heterogeneous *L* x *L* model in SB is fit using

```
options(contrasts=c("contr.treatment","contr.poly"))
fit.hetero<-glm(count~Age*Breath + Status*Age + Age:c.Breath:c.Status, data=table.9.7,
   family=poisson)
summary(fit.hetero, cor=F)

Coefficients:
                      Value Std. Error   t value
        (Intercept) -3.6502502  1.2561249 -2.905961
                Age  5.0922343  1.4323399  3.555186
   BreathBorderline  2.8020966  0.4836257  5.793937
       BreathNormal  6.8009953  0.8208205  8.285606
       StatusFormer  0.2983005  0.1947371  1.531811
        StatusNever  1.1478109  0.3266340  3.514058
AgeBreathBorderline -1.0592502  0.5771326 -1.835367
    AgeBreathNormal -2.0669718  0.9340887 -2.212822
    AgeStatusFormer -1.4076770  0.2297316 -6.127485
     AgeStatusNever -1.0888096  0.3799020 -2.866028
                          Value Std. Error  t value
Age40-50c.Breathc.Status 0.7810585 0.14300764 5.461655
 Age< 40c.Breathc.Status 0.1148594 0.08593526 1.336580

(Dispersion Parameter for Poisson family taken to be 1 )

    Null Deviance: 4097.2 on 17 degrees of freedom

Residual Deviance: 10.80173 on 6 degrees of freedom
```

The last two coefficient estimates give the estimated local log odds ratios for the older and younger groups. The odds ratios for adjacent Smoking and Breathing categories are the same within each age group.

A homogeneous *L* x *L* model is fit using

```
fit.homo<-glm(count~Age*(Breath + Status) + c.Breath:c.Status, data=table.9.7,
family=poisson)
```

If strata are ordered, and a linear trend in log odds ratios is suspected across levels of Z, Agresti fits a model that extends the homogeneous *L* x *L* model to include the term $kI(i = j = 1)\delta$ which adds a multiple of δ to the (1, 1, *k*) cell. The resulting XY log odds ratio at level *k* is then $\beta + kI(i = j = 1)\delta$.

This model is fit to the famous Coal Miners Data, with Age group as strata. We read in the data using

```
table.9.8<-data.frame(expand.grid(Age=factor(c("20-24","25-29","30-34","35-39",
"40-44","45-49","50-54","55-59","60-64"),
levels=c("20-24","25-29","30-34","35-39","40-44","45-49","50-54","55-59","60-64")),
Wheeze=factor(c("Yes","No"), levels=c("Yes", "No")),
Breath=factor(c("Yes","No"), levels=c("Yes", "No")) ),
c.Breath=rep(1:2, each=18), c.Wheeze=rep(rep(1:2,ea=9),2),
extra.term=c(1:9,rep(0,27)),
count=c(9,23,54,121,169,269,404,406,372,7,9,19,48,54,88,117,152,106,95,105,177,257,
273,324,245,225,132,1841,1654,1863,2357,1778,1712,1324,967,526))
```

Note that this data set is available in the R package vcd. Make it available by using

```
library(vcd)
data(CoalMiners)
```

To fit the linear term, we use the following command. I use the label, "extra.term" to stand for δ

```
fit.ord.strata <- glm(count ~ Age * (Breath + Wheeze) + c.Breath:c.Wheeze +
   extra.term, data = table.9.8, family = poisson)
summary(fit.ord.strata, cor = F)
```

```
Coefficients:
                Value Std. Error    t value
(Intercept) -1.2219654  0.2765376  -4.418804
   Age25-29  0.8615296  0.3105417   2.774280
   Age30-34  1.8983149  0.2809356   6.757117
   Age35-39  2.8672944  0.2695779  10.636237
   Age40-44  3.3305489  0.2713944  12.271990
   Age45-49  3.9454944  0.2752234  14.335606
   Age50-54  4.4348682  0.2829411  15.674175
   Age55-59  4.6281967 0.29414802  15.734244
   Age60-64  4.6227534 0.30864611  14.977520
     Breath -1.5893957 0.39165235  -4.058180
     Wheeze -4.3747047 0.40643062 -10.763718
 extra.term -0.1306326 0.02948922  -4.429842

Age25-29Breath -0.7317030 0.31974770  -2.288376
Age30-34Breath -1.2547109 0.29009124  -4.325229
Age35-39Breath -1.8607640  0.2779858  -6.693739
Age40-44Breath -2.2631757  0.2794250  -8.099404
Age45-49Breath -2.7237497  0.2816056  -9.672216
Age50-54Breath -3.4320488  0.2866072 -11.974749
Age55-59Breath -3.8116499  0.2929513 -13.011205
Age60-64Breath -4.2329837  0.3017371 -14.028713
Age25-29Wheeze -0.2386797  0.1416818  -1.684618
Age30-34Wheeze -0.6331931  0.1280614  -4.944451
Age35-39Wheeze -0.7599432  0.1206598  -6.298231
Age40-44Wheeze -1.1026877  0.1200128  -9.188081
Age45-49Wheeze -1.2916855  0.1177208 -10.972452
Age50-54Wheeze -1.3412138  0.1197136 -11.203522
```

```
Age55-59Wheeze -1.4479318  0.1233793 -11.735617
Age60-64Wheeze -1.6556074  0.1337443 -12.378905

c.Breath:c.Wheeze 3.676197  0.1998956 18.39058

(Dispersion Parameter for Poisson family taken to be 1 )

    Null Deviance: 25889.47 on 35 degrees of freedom

Residual Deviance: 6.801743 on 7 degrees of freedom
```

Thus, the estimated *BW* local odds ratio at level *k* of Age is 3.676 – 0.131*k*.

3. Conditional Tests for Ordinal Models in Multi-way Tables

For small samples, the asymptotic chi-squared approximations to goodness-of-fit tests, such as the likelihood-ratio test, do not hold. Instead, theoretically, the p-value for a test of the goodness of fit of a model can be computed from a conditional distribution of the test statistic, by conditioning on estimates of certain nuisance parameters. Conditioning on these sufficient statistics leaves the conditional distribution free of these unknown parameters. So, the p-value is the probability of the test statistic exceeding its observed value, conditional on the sufficient statistic taking on its observed value.

The functions in `exactLoglinTest` give Monte Carlo estimates of conditional p-values for tests of Poisson log-linear models. The function `mcexact` conditions on all sufficient statistics for parameters in a given model, and simulates multi-way tables from the conditional distribution satisfying the observed values of the sufficient statistics. The function approximates the conditional distribution, which is a generalized hypergeometric by default, by sampling from it via either importance sampling (`method = "bab"`) or MCMC (`method = "cab"`). The details of the procedures can found in the citations that are listed in the documentation.

The default test statistics used are the LR test statistic of a model against the saturated model (model deviance) and the Pearson chi-squared statistic.

For example, to fit model (9.12) in Agresti using a conditional test, we use `mcexact` with importance sampling.

```
library(exactLoglinTest)
set.seed(1)

fit.mc<-mcexact(count~Age*(Breath + Wheeze) + c.Breath:c.Wheeze + extra.term,
data=table.9.8, method="bab", nosim=10^4, maxiter = 10^4, savechain=T)
```

The argument `nosim` is the number of simulations desired out of `maxiter` iterations. The reason that `nosim` may not equal `maxiter` is because some simulations from the conditional distribution apparently can result in negative entries in the table. The option `savechain=T` saves the chain of simulations, which can be used to evaluate the importance sampling or MCMC algorithm. `summary` gives more results.

```
summary(fit.mc)

Number of iterations       =  9649
T degrees of freedom       =  3
Number of counts           =  36
df                         =  7
Next update has nosim      =  10000
Next update has maxiter    =  10000
Proportion of valid tables =  0.9649

                deviance    Pearson
observed.stat 6.80174259 6.80829750
```

```
pvalue         0.45862023 0.45123625
mcse           0.01003939 0.00998465
```

"Number of iterations" gives the number of simulations actually obtained. "T degrees of freedom" is a tuning parameter, and df is the model df. "Proportion of valid tables" here is just proportion of simulations done, out of `maxiter`.

In the second half of the output, we have the estimated p-values. We can compare these to their large-sample versions

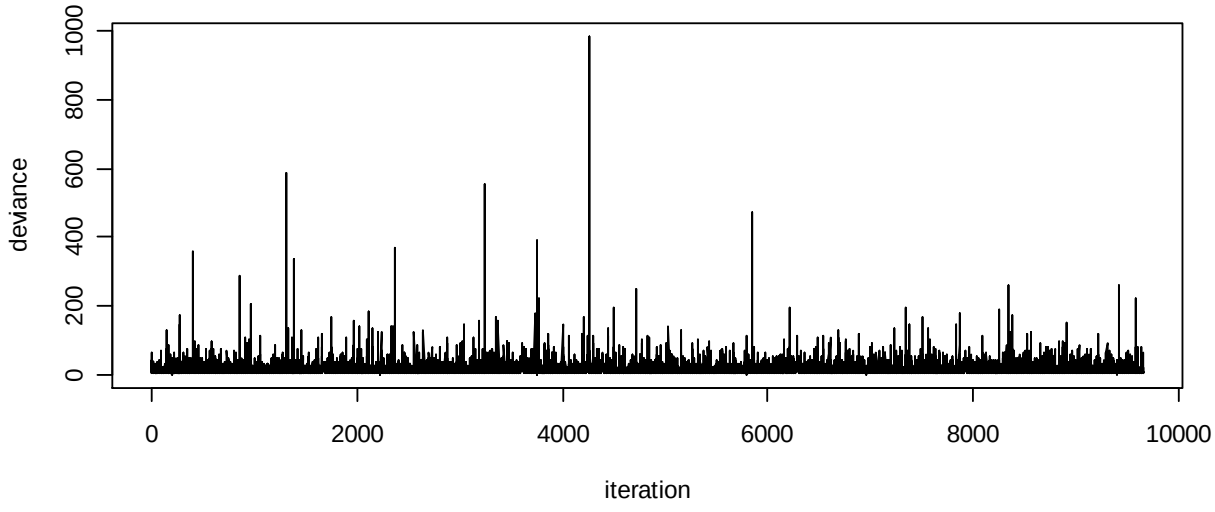
```
pchisq(fit.mc$dobs, df=7, lower=F)
```

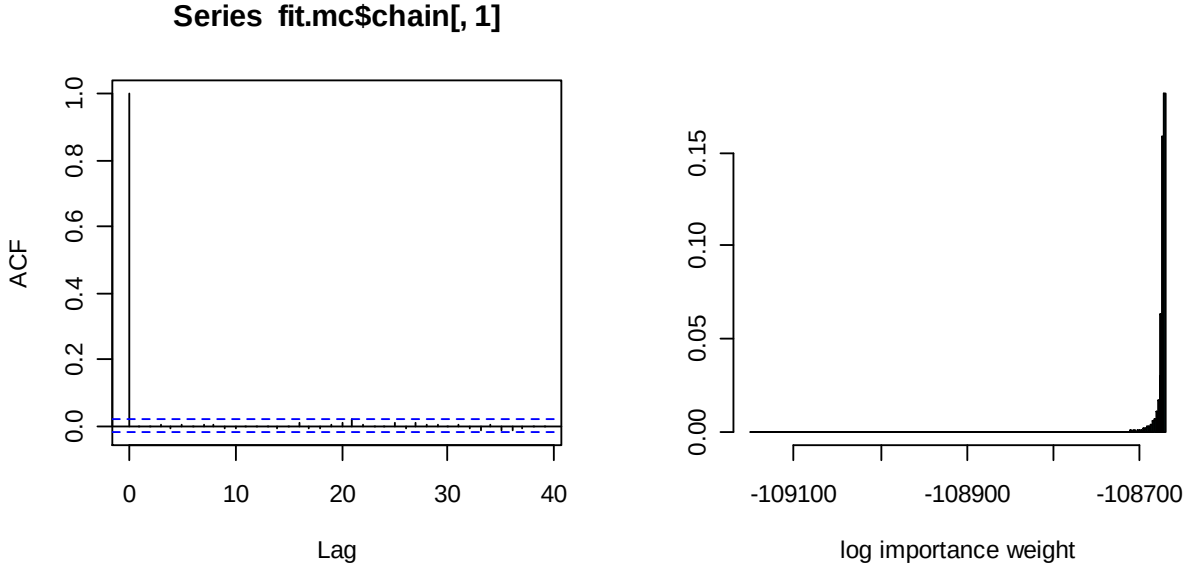
```
[1] 0.4498103 0.4491093
```

They are close, which is expected because of the large sample size.

However, before we "believe" these estimates, we might examine the trace plots and the autocorrelation function of the iterations, as well as the logs of the importance weights (if we use importance sampling). We saved these results by specifying `savechain=T`. They are saved as the `chain` attribute of `fit.mc`.

```
layout(matrix(c(1,1,2,3),2,2, byrow=T))
plot(fit.mc$chain[,1], type="l", xlab="iteration", ylab="deviance")
acf(fit.mc$chain[,1])
library(MASS)
truehist(fit.mc$chain[,3], xlab="log importance weight")
```





The trace plot shows good mixing ("randomness"), and the acf shows low autocorrelations. Also, many of the logs of the importance weights are similar.

To continue the sampling, we can use the `update` method for `bab` (`update.bab`). However, based on the diagnostic plots, and the low Monte Carlo standard errors (in `summary` output) we probably have enough iterations. Large, sparse tables may benefit most from these methods, and we will examine them later in this chapter.

F. Association Models, Correlation Models, and Correspondence Analysis

1. Multiplicative Row and Column Effects Model

The row effects model and the column effects model (as well as the *L* x *L* model) are special cases of the Row-and-column-effects model (Goodman's RC model), which has parameter scores for both rows and columns. The model multiplies the two sets of parameters in the model equation,

$$\log \mu_{ij} = \lambda + \lambda_i^X + \lambda_j^Y + \beta \mu_i \nu_j$$

where μ_i and ν_j are parameter scores. So, instead of having to fix arbitrary scores, we can estimate parameter scores. Agresti notes that finding MLEs is not always easy because the likelihood is not always concave. An iterative modeling fitting algorithm is suggested by Goodman, where one alternates between fitting row effects and column effects models, using the estimated scores from each in turn. In this way, one can ideally get MLEs of the row and column scores. Actually, if corner-point constraints are used for the parameter scores, then one gets the *differences* between each score and the last score (if the last parameter score is set to zero for identifiability). Also, one set of either the row or column parameter score estimates is multiplied by the estimate for β. Thus, the method does not appear to be useful as a way to estimate parameter scores.

In the R package VGAM, there is a function `grc`, which is a front end for the function `rrvglm` (reduced-rank vector generalized linear models), also in the same package. This function will fit an RC model, but not estimate the scores as per Agresti's (9.13) representation of the RC model. Instead, it estimates a matrix of "interaction terms", δ_{ij}, i = 1,..., $I-1$; j = 1,..., $J-1$, where $\delta_{ij} = a_i c_j = \beta(\mu_i - \mu_I)(\nu_j - \nu_J)$, in the language of model (9.13) in Agresti. Separating δ_{ij} into these three components appears to be impossible. However, we can still fit the model, and compare its deviance to that of an L x L model. We will see later that correspondence analysis will give us MLEs of the row and column scores.

When I prepare the data for `grc`, I use "reverse" levels for the factors so that the last category of the respective factor is zeroed out instead of the first category.

```
table.9.9<-data.frame(expand.grid(MH=factor(c("well","mild","moderate","impaired"),
levels=rev(c("well","mild","moderate","impaired"))),
SES=factor(LETTERS[1:6], levels=LETTERS[6:1])),
count=c(64,94,58,46,57,94,54,40,57,105,65,60,71,141,77,94,36,97,54,78,21,71,54,71))
```

`grc` requires the data to be in `array` or `table` format. So, I transform it to a table.

```
table.9.9.matrix<-t(xtabs(count~MH + SES ,data = table.9.9))       # R only
```

Now, I fit the RC model. The `Rank` is equivalent to Agresti's *M** on p. 380, and `Rank = 1` will result in model (9.13). (By the way, there is full documentation for VGAM on the author's website and corresponding papers. The remaining defaults are explained there.)

```
library(vgam)
options(contrasts=c("contr.treatment", "contr.poly"))
(fit.rc<-grc(table.9.9.matrix, Rank=1))

Call:
rrvglm(formula = as.formula(str2), family = poissonff, data = .grc.df,
    control = myrrcontrol, constraints = cms)

Coefficients:
 (Intercept)         Row2         Row3         Row4         Row5         Row6
 4.298506527  0.040277163  0.197234822 -0.193454278 -0.543859296 -0.474304500


        Col2         Col3         Col4            E            D            C
-0.432208305 -0.006267693 -1.184327467  0.172181348  0.376512091  0.467578503

           B            A
 0.628055583  0.626074261

Residual Deviance: 3.480503 on 8 degrees of freedom
Log-likelihood: -73.81976 on 8 degrees of freedom
```

In the output, the Row and Column coefficients refer to the λ_i^X and λ_j^Y's. Because we set the last category to zero, Row2 refers to SES E and Row6 refers to SES A. Also, Col2 refers to Moderate, and Col4 refers to Well. The coefficients labeled A through E refer to the first five c_j values (the last is 0). A call to `summary` (i.e., `summary(fit.rc)`) gives some more information, as well as standard errors.

To get the matrix of δ_{ij}'s, we first compute a biplot (without plotting it). The use of `biplot` reflects the fact that the c_j values are involved in a latent variable interpretation in the `rrvglm` context. However, we only use `biplot` here to get the interaction term estimates.

```
res<-biplot(fit.rc, plot.it=F)
res$Cmatrix
res$Amatrix
```

The matrix of δ_{ij}'s is then

```
Delta<-structure(rbind(0,res$Cmatrix)%*%t(res$Amatrix),
dimnames=list(c(LETTERS[6:1]),c(rev(c("well","mild","moderate","impaired")) )))

Delta
```

```
  impaired  moderate      mild      well
F        0 0.0000000 0.0000000 0.0000000
E        0 0.1721813 0.2088179 0.4150163
D        0 0.3765121 0.4566259 0.9075237
C        0 0.4675785 0.5670694 1.1270250
B        0 0.6280556 0.7616926 1.5138300
A        0 0.6260743 0.7592897 1.5090543
```

Now, I show that these values are the products $\beta(\mu_i - \mu_I)(\nu_j - \nu_J)$, within some rounding error.

```
mu<-c(-1.68, -.14, .14,1.41)      # Agresti's scores
nu<-c(-1.11,-1.12,-.37,.03,1.01,1.82)
mu<-mu-mu[4]         # take differences with last category
nu<-nu-nu[6]

structure(t(0.17*mu%*%t(nu)),
dimnames=list(c(LETTERS[1:6]),c("well","mild","moderate","impaired") ) )
```

```
      well     mild moderate impaired
A 1.539129 0.772055 0.632587        0
B 1.544382 0.774690 0.634746        0
C 1.150407 0.577065 0.472821        0
D 0.940287 0.471665 0.386461        0
E 0.425493 0.213435 0.174879        0
F 0.000000 0.000000 0.000000        0
```

However, working backwards from `Delta` to its component scores is what is desired.

To compare with an *L* x *L* model with equal interval scores, we fit this model by adding fixed scores (`c.SES` and `c.MH`) to the data frame (you could also put them directly into the formula in `glm`).

```
table.9.9a<-data.frame(expand.grid(MH=factor(c("well","mild","moderate","impaired"),
levels=rev(c("well","mild","moderate","impaired"))),
SES=factor(LETTERS[1:6], levels=LETTERS[6:1])),
c.MH=rep(4:1,6), c.SES=rep(6:1,each=4),
count=c(64,94,58,46,57,94,54,40,57,105,65,60,71,141,77,94,36,97,54,78,21,71,54,71))
```

```
(fit.LL<-glm(count~MH + SES + c.MH:c.SES, family=poisson,data=table.9.9a))

Coefficients:
(Intercept)   MHmoderate       MHmild       MHwell          SESE          SESD
   4.141375    -0.370975    -0.182468    -1.200474    -0.009291     0.140934
       SESC          SESB         SESA   c.MH:c.SES
  -0.374645    -0.768734    -0.946158     0.090929

Degrees of Freedom: 23 Total (i.e. Null);  14 Residual
Null Deviance:      217.3
Residual Deviance: 9.707        AIC: 173.9
```

A LR test of the parameter scores versus equal interval scores is

```
fit.LL$deviance-fit.rc@criterion$deviance

[1] 6.226855
```

with 14 – 8 = 6 df. Since

```
pchisq(6.226855,df=6, lower.tail=F)

[1] 0.3982635
```

the RC model does not provide a significantly better fit.

2. Canonical Correlation Model

A canonical correlation model is also an association model. It fits the joint probabilities

$$\pi_{ij} = \pi_{i+}\pi_{j+}\left(1+\sum_{k=1}^{M}\lambda_k\mu_{ik}\nu_{jk}\right) \tag{9.1}$$

where M = min(I – 1, J – 1), λ_k is the correlation between scores $\{\mu_{ik}, i = 1,\ldots, I\}$ and $\{\nu_{jk}, j = 1,\ldots, J\}$, and the scores $\{\mu_{ik}, i = 1,\ldots, I\}$ and $\{\nu_{jk}, j = 1,\ldots, J\}$ maximize the correlation λ_k (subject to μ_{ik} and μ_{ik-1} being uncorrelated, and ν_{jk} and ν_{jk-1} being uncorrelated, if applicable). If the λ_k are zero, the model reduces to the independence model.

We can fit a one-dimensional (M = 1) canonical correlation model using the `corresp` function from the MASS library. Agresti notes that when λ in (9.1) is close to zero, the MLEs of λ and the score parameters are similar to those of β and the score parameters from the RC model. In fact, we can check this because a one-dimensional fit shows that the MLE of λ is small.

First, I define a new data frame called table.9.9b to have only the MH and SES factors, and to put the levels back in "forward" order.

```
table.9.9b<-data.frame(expand.grid(MH=factor(c("well","mild","moderate","impaired"),
   levels=(c("well","mild","moderate","impaired"))),
SES=factor(LETTERS[1:6], levels=(LETTERS[1:6]))),
count=c(64,94,58,46,57,94,54,40,57,105,65,60,71,141,77,94,36,97,54,78,21,71,54,71))
```

Now, I use `corresp` with number of factors, `nf` = 1.

```
corresp(x=design.table(table.9.9b), nf=1)
# R:  corresp(x=xtabs(count~MH+SES,data=table.9.9b))
```

```
First canonical correlation(s): 0.1615119

 MH scores:
     well       mild   moderate impaired
 -1.611765 -0.1822571 0.08506316 1.470761

 SES scores:
        A         B          C          D        E        F
 -1.127146 -1.151843 -0.3717666 0.07545902 1.018916 1.775564
```

Compare these estimates with those on p. 381 in Agresti.

We could fit a saturated model (i.e., $M = 3$) by using `nf = 3` or by using functions for correspondence analysis, described next.

3. Correspondence Analysis

Correspondence analysis is a graphical method for describing associations among categorical variables. The rows and columns of a contingency table are represented by points on a graph. The greater the magnitude of the projections of the points onto an axis of the graph, the greater the association described by that axis. Using the same notation as for equation (9.1) above, correspondence analysis uses the adjusted scores

$$x_{ik} = \lambda_k \mu_{ik}, \qquad y_{jk} = \lambda_k \nu_{jk}$$

Then, the graph of the first two dimensions plots (x_{i1}, x_{i2}) for each row, and (y_{j1}, y_{j2}) for each column.

There are many functions for doing correspondence analysis in S-PLUS and R. I will try to illustrate different aspects of each.

The library `multiv` in both S-PLUS and R has function `ca` for doing correspondence analysis and function `plaxes` to help with creating a plot like that in Figure 9.4 of Agresti. It needs an array or table, so

```
table.9.9b.array<-t(design.table(table.9.9b))  # transposed to match table 9.10
# R:  table.9.9b.array<-t(xtabs(count~MH+SES,data=table.9.9b))

library(multiv)
(fit.ca<-ca(table.9.9b.array, nf=3))

$evals:
[1] 0.0260860808 0.0013648955 0.0002818488

$rproj:
       "Factor1"   "Factor2"    "Factor3"
[1,]  0.18204759 -0.01965218  0.027711980
[2,]  0.18603645 -0.01028829 -0.026940068
[3,]  0.06004474 -0.02157959 -0.010481182
[4,] -0.01218750  0.04121665  0.009748941
[5,] -0.16456708  0.04381706 -0.008189913
[6,] -0.28677478 -0.06237160  0.003613989

$cproj:
       "Factor1"   "Factor2"    "Factor3"
[1,]  0.26031947  0.01059579  0.022151845
[2,]  0.02943694  0.02487137 -0.018917788
[3,] -0.01373865 -0.06926409 -0.004141493
[4,] -0.23754551  0.01763209  0.015692677
```

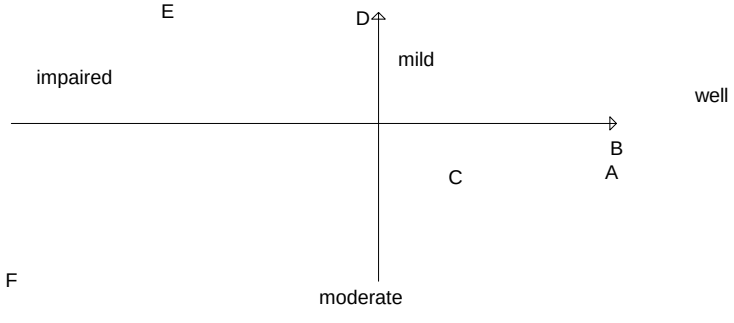
`evals` are the squared correlations, the λ_k above. The second and third are almost zero, showing lesser importance of the second and third dimensions in explaining variability. `rproj` and `cproj` are the row and column scores on each of the three dimensions.

To plot the scores, use

```
# plot of first and second factors
plot(fit.ca$rproj[,1], fit.ca$rproj[,2],type="n",ylim=c(-.1,.1),xlim=c(-.3,.3),
   xlab="",ylab="",axes=F)
text(fit.ca$rproj[,1], fit.ca$rproj[,2], labels=dimnames(table.9.9b.array)$SES)
text(fit.ca$cproj[,1], fit.ca$cproj[,2], labels=dimnames(table.9.9b.array)$MH)
# Place additional axes through x=0 and y=0:
my.plaxes(fit.ca$rproj[,1], fit.ca$rproj[,2],size=.15)
# R: my.plaxes.f(fit.ca$rproj[,1], fit.ca$rproj[,2],Length=.15)
```

where `my.plaxes` is a modification of `plaxes`

```
# S-PLUS
my.plaxes<- function(a, b, size = 0.1)
{
   arrows(min(a), 0, max(a), 0, size = size)
   arrows(0, min(b), 0, max(b), size = size)
}

# R
my.plaxes<-function(a, b, Length = 0.1)
{
    arrows(min(a), 0, max(a), 0, length = Length)
    arrows(0, min(b), 0, max(b), length = Length)
}
```



Another way to do correspondence analysis is via the `corresp` function from the MASS library. However, this time we set the number of factors (`nf`) to 3.

```
fit.corresp<-corresp(x=design.table(table.9.9b),nf=3)
```

The scores are scaled, so we must re-scale by multiplying by the canonical correlations (`fit.corresp$cor`)

```
fit.corresp$cor
[1] 0.16151194 0.03694460 0.01678778

fit.corresp$rscore %*% diag(fit.corresp$cor)
                [,1]        [,2]         [,3]
    well -0.26031933 -0.01059603  0.022154293
    mild -0.02943669 -0.02487118 -0.018914368
moderate  0.01373872  0.06926415 -0.004136768
impaired  0.23754547 -0.01763185  0.015693435
```

```
fit.corresp$cscore %*% diag(fit.corresp$cor)
          [,1]         [,2]          [,3]
A -0.18204759  0.01965179  0.027712258
B -0.18603645  0.01028853 -0.026939901
C -0.06004474  0.02157969 -0.010480869
D  0.01218753 -0.04121676  0.009748570
E  0.16456713 -0.04381688 -0.008190311
F  0.28677477  0.06237172  0.003614761
```

Plotting is done via `plot(fit.corresp)`

The R package `CoCoAn` will do (constrained) correspondence analysis via the function `CAIV`. The call takes an array (table) and here would be

```
library(CoCoAn)
fit.CAIV<-CAIV(table.9.9b.array)
```

Finally, a generalization of correspondence analysis to multi-way tables is given in the R function `FCAk` from package `PTAk`.

G. Poisson Regression for Rates

To model the rate of occurrence of an event over an exposure time, dependent on covariates, a Poisson loglinear model is useful. If the response count for the *i*th individual is n_i over exposure time t_i, then the expected rate is μ_i / t_i, the log of which is modeled is a linear function of covariates

$$\log(\mu_i / t_i) = \alpha + \beta x_i$$

The term $\log t_i$ is then considered an offset.

For this section, Agresti uses the data in Table 9.1 - Heart valve operations. Patients were classified by type of heart valve (aortic, mitral) and age (< 55, 55+). Follow-up observation lasted from the time of operation until the patient died or the study ended. The follow-up observation for each patient is their time at risk, in months. For each cell in the 2 x 2 table, the total time at risk is the sum of the times at risk for patients in that cell. Also recorded is the number of deaths per cell. The sample death rate is then the number of deaths divided by the total time at risk.

After I read in the data, I use the function `glm` to fit a loglinear model to the responses because I want to include the offset term, exposure.

```
table.9.11<-data.frame(expand.grid(factor(c("Aortic","Mitral")),factor(c("<55","55+"),
   levels= c("<55","55+"))), Deaths=c(4,1,7,9), Exposure=c(1259,2082,1417,1647))
names(table.9.11)[1:2]<-c("Valve","Age")
attach(table.9.11)
(table.9.11<-data.frame(table.9.11,Risk=Deaths/Exposure))

   Valve Age Deaths Exposure         Risk
1 Aortic <55      4     1259 0.0031771247
2 Mitral <55      1     2082 0.0004803074
3 Aortic 55+      7     1417 0.0049400141
4 Mitral 55+      9     1647 0.0054644809


options(contrasts=c("contr.treatment", "contr.poly"))
fit.rate<-glm(Deaths~Valve+Age+offset(log(Exposure)),family=poisson,data=table.9.11)

summary(fit.rate)
```

```
Coefficients:
                Value Std. Error    t value
(Intercept) -6.3120972  0.5064590 -12.463196
      Valve -0.3298665  0.4381267  -0.752902
        Age  1.2209479  0.5136586   2.376964

(Dispersion Parameter for Poisson family taken to be 1 )

    Null Deviance: 10.84053 on 3 degrees of freedom

Residual Deviance: 3.222511 on 1 degrees of freedom

Number of Fisher Scoring Iterations: 4
```

Thus, the estimated rate for the older age group (coded 1, by the ordering of the levels) is exp(1.221) = 3.4 times that for the younger group. The LR confidence intervals are

```
library(MASS)
exp(confint(fit.rate))
```

```
                    2.5 %       97.5 %
(Intercept) 0.0005868836  0.004396417
      Valve 0.2988564722  1.709356104
        Age 1.3239556759 10.392076295
```

The chi-squared statistic is

```
sum(resid(fit.rate, type = "pearson")^2)
[1] 3.113503
```

which is *not* a significantly poor fit at the .05 level.

Fitted values are obtain by

```
attach(table.9.11)
mhat<-fitted(fit.rate)
exphat<-fitted(fit.rate)/Exposure
temp<-rbind(mhat,exphat)
array(temp,dim=c(2,2,2),dimnames=list(c("Deaths","Risk"),Valve=c("Aortic","Mitral"),Ag
    e=c("<55","55+")))
```

```
, , <55
            Aortic      Mitral
Deaths 2.284108702 2.715892496
  Risk 0.001814225 0.001304463

, , 55+
            Aortic      Mitral
Deaths 8.715891783 7.284108228
  Risk 0.006150947 0.004422652
```

The model with identity link can be fit by including terms that are products of the numerical codes for Valve and Age with Exposure, and by removing an intercept term. To get the product of Valve and Exposure, we extract the numerical codes from Valve and multiply by Exposure, placing all this in an `I()` function (*as is* function). For some reason, the codes for Age are backwards (counting down from <55 instead of up), so I reversed the codes to get the equivalent of older = 1 and younger = 0. The difference is that the sign of the coefficient is now positive instead of negative.

```
fit.id<-glm(Deaths~I(codes(Valve)*Exposure) + I(rev(codes(Age))*Exposure) + Exposure-
    1,family=poisson(link=identity),data=table.9.11)
```

```
summary(fit.id, cor=F)

Coefficients:
                                    Value  Std. Error    t value
   I(codes(Valve) * Exposure) -0.001936109 0.001315309 -1.4719796
I(rev(codes(Age)) * Exposure)  0.003965548 0.001440062  2.7537340
                     Exposure  0.000479430 0.003249681  0.1475314

(Dispersion Parameter for Poisson family taken to be 1 )

    Null Deviance: Inf on 4 degrees of freedom

Residual Deviance: 1.093067 on 1 degrees of freedom

Number of Fisher Scoring Iterations: 4
```

H. Modeling Survival Times

Agresti notes the connection between ML estimation using a Poisson likelihood and a survival-time likelihood using an exponential hazard function. So, we can use methods for fitting loglinear models to fit these types of survival models.

The random variable is the time to some event. For the *i*th subject, this is t_i. If the *i*th subject does not experience the event during the study time, then their recorded observation is censored. Let w_i = 0 if the *i*th observation is censored, and 1 otherwise. Then, if the exponential hazard is modeled as

$$h(t;\mathbf{x}) = \lambda \exp(\boldsymbol{\beta}'\mathbf{x})$$

for covariates in $\mathbf{x}$, then the survival-time log likelihood as a function of $\boldsymbol{\beta}$ is identical to the log likelihood for independent Poisson variates w_i with expected values $\mu_i = t_i\, h(t_i;\mathbf{x}_i)$. Thus, we fit the Poisson loglinear model $\log(\mu_i/t_i) = \log\lambda + \boldsymbol{\beta}'\mathbf{x}_i$ to responses $\{w_i\}$.

In the Lung Cancer Survival example, Agresti uses a piecewise constant hazard rate instead of a constant hazard rate. The hazard is constant in two-month intervals of follow-up time. The data contain counts of deaths within each follow-up interval, by factors Histology (I, II, III) and Stage of disease (1, 2, 3). When we read in the data set, the variable Deaths is the number of deaths at each cell of the table, or the sum of the $\{w_{ijk}\}$ for the *i*th histology, *j*th stage, and *k*th time interval.

As usual, we reverse the levels of factors so that the `glm` coding sets the *last* category to zero instead of the first.

```
table.9.13<-expand.grid(Stage=factor(c(1,2,3),
   levels=3:1),Histology=factor(c("I","II","III"), levels=rev(c("I","II","III"))),
Time=factor(c(0,2,4,6,8,10,12), levels=rev(c(0,2,4,6,8,10,12))))
table.9.13<-data.frame(table.9.13,
   Deaths=c(9,12,42,5,4,28,1,1,19,2,7,26,2,3,19,1,1,11, 9,5,12,3,5,10,1,3,7,
10,10,10,2,4,5,1,1,6,1,4,5,2,2,0,0,0,3,3,3,4,2,1,3,1,0,3,1,4,1,2,4,2,0,2,3),
Exposure=c(157,134,212,77,71,130,21,22,101,139,110,136,68,63,72,17,18,63,126,96,90,63,
   58,42,14,14,43,102,86,64, 55,42,21,12,10,32,88,66,47,50,35,14, 10,8,21,82,
   59,39,45,32,13,8,8,14,76,51,29,42,28,7,6,6,10) )
```

Now, I fit a main effects model,

```
options(contrasts=c("contr.treatment", "contr.poly"))
```

```
fit.surv<-glm(Deaths~Histology+Time+Stage+offset(log(Exposure)), data=table.9.13,
   family=poisson,link=log)
summary(fit.surv, cor=F)

Coefficients:
                 Value Std. Error    t value
(Intercept) -1.75261448  0.2626080 -6.6738807
HistologyII  0.05490011  0.1599783  0.3431722
 HistologyI -0.10754218  0.1474495 -0.7293492
     Time10 -0.17497139  0.3203716 -0.5461515
      Time8 -0.48992699  0.3339600 -1.4670228
      Time6  0.29410425  0.2706762  1.0865538
      Time4  0.09545150  0.2669593  0.3575507
      Time2  0.04772845  0.2596595  0.1838117
      Time0  0.17518255  0.2497592  0.7014058
     Stage2 -0.85429714  0.1368623 -6.2420187
     Stage1 -1.32431049  0.1520463 -8.7099133

(Dispersion Parameter for Poisson family taken to be 1 )

    Null Deviance: 175.7178 on 62 degrees of freedom

Residual Deviance: 43.92253 on 52 degrees of freedom

Number of Fisher Scoring Iterations: 4
```

Note that we get the estimates of the effects of stage of disease shown on p. 390 of Agresti:

-0.85 + 1.32 = 0.47 (stage 2 – stage 1)
0 + 1.32 = 1.32 (stage 3 – stage 1)

so that at fixed follow-up time and histology, the estimated death rate at the second disease stage is exp(0.47) = 1.60 times that at the first stage.

As Agresti notes, as long as neither of the covariates (Stage or Histology) interacts with Time Interval, we have a proportional hazards model. That is, the hazards at two different covariate combinations are proportional to one another (proportionality constant independent of time).

One can use stepAIC in the MASS library to find a model with lowest AIC. Here is the final result using as "upper scope" a model with all two-way interactions, and lower scope a model with all main effects except Histology.

```
fit2<-glm(Deaths~Time+offset(log(Exposure)),data=table.9.13,family=poisson)
library(MASS)
stepAIC(fit.surv,scope=list(lower=formula(fit2),upper=~.^2),direction="both",
   trace=F)$anova

Stepwise Model Path
Analysis of Deviance Table

Initial Model:
Deaths ~ Histology + Time + Stage + offset(log(Exposure))

Final Model:
Deaths ~ Time + Stage + offset(log(Exposure))


         Step Df Deviance Resid. Df Resid. Dev      AIC
1                              52   43.92253 65.92253
2 - Histology  2 1.876473        54   45.79901 63.79901
```

A model without Histology has the lowest AIC.

I. Empty Cells and Sparseness

When some cells in a table are empty, ordinary ML estimation may give infinite estimates of model parameters, or very large estimates with large standard errors. Also, goodness-of-fit statistics may not have asymptotic chi-squared distributions. Alternative tests include exact small-sample tests and Monte Carlo approximations to exact tests.

For example, for the pharmaceutical clinical trials example, with five centers, two centers have no successes for either the treatment or placebo groups. A logit model with factors Treatment and Center, with no intercept gives nonsense estimates for the Center 1 and 3 effects because the Center-Response marginal table has zeroes.

```
table.9.16<-data.frame(expand.grid(treatment=c(1,0), Center= factor(1:5, levels=1:5)),
prop=c(0,0,1/13,0,0,0,6/9,2/8,5/14,2/14))

options(contrasts=c("contr.treatment", "contr.poly"))
fit.glm.sparse<-glm(prop~treatment+Center-1, data=table.9.16, weights=c(5,9,13,10,7,
5,9,8,14,14), family=binomial)

Warning message:
Algorithm did not converge in: (if (is.empty.model(mt)) glm.fit.null else glm.fit)(x =
X, y = Y,

summary(fit.glm.sparse)    # R output

Coefficients:
          Estimate Std. Error z value Pr(>|z|)
treatment   1.5460     0.7016   2.203 0.027568 *
Center1   -13.5922    95.2086  -0.143 0.886478
Center2    -4.2025     1.1891  -3.534 0.000409 ***
Center3   -13.5874    87.9646  -0.154 0.877244
Center4    -0.9592     0.6548  -1.465 0.142955
Center5    -2.0223     0.6700  -3.019 0.002540 **
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 73.07369  on 10  degrees of freedom
Residual deviance:  0.50231  on  4  degrees of freedom
AIC: 24.859

Number of Fisher Scoring iterations: 10
```

An approximate likelihood-ratio test of the treatment effect compares the deviance between the fit above and the fit without `treatment`.

```
fit.glm.sparse2<-glm(prop~Center-1, data=table.9.16, weights=c(5,9,13,10,7,5,9,8, 14,
14), family=binomial)

anova(fit.glm.sparse2,fit.glm.sparse)   # R output

Analysis of Deviance Table

Model 1: prop ~ Center - 1
Model 2: prop ~ treatment + Center - 1
  Resid. Df Resid. Dev Df Deviance
1         5     5.9880
2         4     0.5023  1   5.4856      # LRT
```

However, we can use the `survival` library to get an exact likelihood ratio test of the treatment effect. Recall that to use `clogit` (R) or `coxph` (S-PLUS) for conditional logistic regression, the data must be in 0's and 1's (i.e., success, failure) instead of counts out of totals. I modify the data first, and rename the table `table.9.16a`. I call the success/failure factor, `case`.

```
temp<-length(20)
temp[rep(c(T,F),10)]<-c(0,0,1,0,0,0,6,2,5,2)          # 1
temp[rep(c(F,T),10)]<-c(5,9,12,10,7,5,3,6,9,12)       # 0
table.9.16a<-data.frame(table.9.16[rep(1:10,c(5,9,13,10,7,5,9,8,14,14)),1:2],
case=rep(rep(c(1,0),10),temp))
```

Now, I call `coxph` (or it's R wrapper, `clogit`) using `Center` as a strata variable, with `method="exact"`.

```
library(survival)   # R only
clogit(case~treatment+strata(Center), data=table.9.16a, method="exact")
# S-PLUS: coxph(Surv(rep(1,sum(temp)),case)~treatment + strata(Center),
   data=table.9.16a, method="exact")


          coef exp(coef) se(coef)    z     p
treatment 1.47      4.35    0.682 2.16 0.031

Likelihood ratio test=5.23  on 1 df, p=0.0222  n= 94
```

Thus, there is a significant treatment effect.

Chapter 10 – Models for Matched Pairs

A. Summary of Chapter 10, Agresti

Matched pairs data contain observations on two response variables where each observation from one response variable pairs with one and only one observation from the other response variable. Thus, the two response variables are dependent. This chapter examines analyses of matched-pairs data with categorical responses.

The first section considers binary outcomes. *Marginal homogeneity* occurs when the marginal probabilities for one response equal the corresponding marginal probabilities for the other response. This implies that the off-diagonal joint probabilities are also equal (*symmetry*). Mcnemar's Test can be used to compare the corresponding dependent proportions.

"Subject-specific" tables have separate tables for each matched pair. They allow probabilities to vary by pair. (Thus, a better term might be *pair*-specific tables). Population-averaged tables sum the subject-specific tables and correspond to marginal models. Subject-specific tables are used to fit logit models that allow each pair to have their own probability distribution. However, with these models, there are usually more parameters than pairs, leading to problems with ML estimation. Solutions include conditional ML estimation and random effects models.

Extensions to matched-pairs with multinomial responses and additional explanatory variables use the random effects approach. This is because the conditional ML approach can only estimate within-pair or within-cluster effects. Between-cluster effects (covariates that are constant within any given cluster) will cancel out of the conditional likelihood. Marginal models for matched-pairs for multinomial responses are discussed in Section 10.3.

Instead of modeling the marginal probabilities, one can model the joint cell probabilities. Cell probabilities that satisfy *symmetry* (i.e., $\pi_{ab} = \pi_{ba}$ for $a \neq b$) also satisfy marginal homogeneity, but not necessarily vice versa (unless quasi-symmetry also holds). The equivalent loglinear model has pairwise association terms independent of the order of the subscripts (e.g., $\lambda_{ab} = \lambda_{ba}$ for $a \neq b$). Because of its simplicity, a symmetry model fits well in very few applications. A *quasi-symmetry* loglinear model permits the main-effect terms in the symmetry model to differ, so that $\lambda_{ab} = \lambda_{ba}$ for $\log \mu_{ab}$ and $\log \mu_{ba}$, but the main-effect terms differ so that $\log \mu_{ab} \neq \log \mu_{ba}$, as it is for symmetry (see equation 10.19 in Agresti). The differences in the MLEs of the main effects parameters $\left\{\lambda_j^Y - \lambda_j^X\right\}$ are equal to conditional MLEs of the $\left\{\beta_j\right\}$ in a baseline-category logit model applied to a population-averaged table. Finally, a square table satisfies *quasi-independence* if the row and column variables are independent given that the row and column outcomes differ. Ordinal versions of quasi-symmetry and quasi-independence are given in Sections 10.4.6 – 10.4.8.

Matched-pairs models are used to analyze agreement between two observers. An independence model assumes no agreement. Quasi-independence and quasi-symmetry models imply some association. A numerical index of agreement is Cohen's Kappa, ranging between 0 = agreement by chance only and 1 = perfect agreement.

Paired preferences are common in analyses using preference data. The data may consist of an I x I table whose cells contain the number of preferences for the column designation versus the row designation. The Bradley-Terry logit model can be used to model influences on the probability that one member is preferred to another. The model can be extended to handle comparisons on ordinal scales.

Matched-sets are the generalization of matched-pairs to three or more responses. There are analogs for marginal homogeneity, symmetry, and quasi-symmetry for matched-sets. Marginal homogeneity is the equality of marginal probabilities in the multi-way table. Symmetry is the equality of symmetric joint probabilities, where symmetric means permuted responses. The log-linear quasi-symmetry model is the symmetry model, plus factors representing main effect terms for the marginals.

B. Comparing Dependent Proportions

In a 2 x 2 table, if π_{1+} denotes the marginal "success" probability for response 1, and π_{+1} denotes the marginal "success" probability for response 2, then to test for marginal homogeneity ($H_0 : \pi_{1+} = \pi_{+1}$), one can use the difference between the corresponding sample proportions. The square of the score statistic for this test (equation 10.4 in Agresti) is the test statistic of Mcnemar's Test, which has chi-squared distribution with 1 df. Mcnemar's test does not use the counts on the main diagonal because they are irrelevant to decide whether the marginal probabilities π_{1+} and π_{+1} differ. Confidence intervals on the difference ($\pi_{1+} - \pi_{1+}$), based on large-sample theory, do depend on the main diagonal counts.

The Prime Minister Approval Rating example compares the proportion of survey respondents who approved of the UK Prime Minister on two different occasions (the responses).

```
table.10.1<-matrix(c(794,150,86,570),byrow=T,ncol=2)
```

Mcnemar's test is built into both R and S-PLUS. This tests the hypothesis of marginal homogeneity.

```
mcnemar.test(table.10.1,correct=F)
```

```
   McNemar's chi-square test without continuity correction

data:  table.10.1
McNemar's chi-square = 17.3559, df = 1, p-value = 0
```

Marginal homogeneity is rejected. A confidence interval on the difference between the two marginal probabilities is given using a large-sample formula in (10.2) of Agresti. The computation is aided by two utility functions in the base library of R: `marginal.table` and `prop.table`. Both of these functions can be sourced directly into S-PLUS (but without the "environment" information at the bottom).

`prop.table` gives the table of sample proportions.

```
table.10.1.prop<-prop.table(table.10.1)
```

`marginal.table` gives the marginal sums for rows (1) or columns (2). I use it to get d.

```
prop.diff<-margin.table(table.10.1.prop,2)[1]-margin.table(table.10.1.prop,1)[1]
```

Next, the off-diagonal proportions are used in the computation of the standard error.

```
off.diag<-diag(table.10.1.prop[1:2,2:1])
```

Here is the 95% confidence interval

```
prop.diff + c(-1,1)*qnorm(.975)*sqrt((sum(off.diag) - diff(off.diag)^2) /
sum(table.10.1))
```

```
[1] -0.05871612 -0.02128388
```

It shows that the probability of approval decreases between 2 and 6 percent across the two occasions.

A small-sample test comparing dependent proportions conditions on the sum of the off-diagonal counts. Then, one of the two terms in the sum is binomially distributed with index equal to this sum, and probability ½.

Also, McNemar's test is equivalent to a Cochran-Mantel-Haenszel test with subjects as strata. Thus, the CMH test is performed on a 2 x 2 x n table. This table is called the subject-specific table, and will be used in the next section.

C. Conditional Logistic Regression for Binary Matched Pairs

A subject-specific table (or, more accurately, a pair-specific table) can be used to fit a logit model with pair-specific probabilities. Specifically, for binary response Y_{it} from the *t*th observation from the *i*th pair

$$\text{logit}\left[P\left(Y_{it}=1\right)\right]=\alpha_i+\beta x_t$$

where $x_1=0$ and $x_2=1$. With this model, for each pair, the odds of success for observation 2 are $\exp(\beta)$ times the odds for observation 1, and this odds ratio is common across pairs. Conditional on the α_i and β, pairs are independent of each other, and within a pair, observations are independent of each other. This means that the association within a pair is described completely by α_i, and pairs are only associated via the common effect, β.

Conditional logistic regression can be used to find the MLE for β, where one conditions on sufficient statistics for the α_i, which are the pairwise success totals $\{S_i=y_{i1}+y_{i2}\}$ (see section 10.2.3 in Agresti). This distribution only depends on β when at least one of the y_{it} is 1. Thus, inference on β only depends on outcomes in different categories at the two observations.

An alternative way to find the MLE for β is to treat the α_i as random effects, with identical normal distributions. Integrating the likelihood with respect to the distributions of the α_i yields a marginal likelihood for β, which is maximized to get the MLE. The model is called a generalized linear mixed model. The MLE is the same for either the conditional ML estimation or ML estimation via the generalized linear mixed model.

Agresti illustrates a case-control study where 144 cases have myocardial infarction (MI) and 144 controls, matched to cases by age and gender, do not. Thus, there are 144 pairs. Each member of a pair was also asked whether they had diabetes (1 = yes, 0 = no). The response in this model is MI, which is fixed. The predictor is diabetes, which is random. The study is thus retrospective, but we can still get an estimate of the XY odds ratio, $\exp(\beta)$ in a logistic regression model, by using the fixed responses (see chapter 2). The model for subject *t* in pair *i* is

$$\text{logit}\left[P\left(Y_{it}=1\right)\right]=\alpha_i+\beta x_{it}$$

We can get the MLE of β using conditional logistic regression or by integrating out random effects. The population-averaged table is in Table 10.3 in Agresti. The pairs for the subject-specific table are in Table 10.4. We input the data into S using the subject-specific form. The following data frame has the variables: pair (either subject 1 or subject 2 of the matched pair), MI (myocardial infarction; yes=1, no=0), diabetes (yes=1, no=0).

```
table.10.3<-data.frame(pair=rep(1:144,rep(2,144)), MI=rep(c(0,1),144),
   diabetes=c(rep(c(1,1),9),rep(c(1,0),16),rep(c(0,1),37),rep(c(0,0),82))
)
```

For a conditional logistic regression, we can use `coxph` with `method=exact`, as we did in Section J of Chapter 6. Recall that for R, we use the front-end function `clogit`.

```
fit.CLR<-coxph(Surv(rep(1,2*144),MI)~diabetes+strata(pair),method="exact",
   data=table.10.3)
# R: library(survival)
# R: fit.CLR<-clogit(MI~diabetes+strata(pair),method="exact", data=table.10.3)
```

```
summary(fit.CLR)

  n= 288

          coef exp(coef) se(coef)    z      p
diabetes 0.838      2.31    0.299 2.8 0.0051

         exp(coef) exp(-coef) lower .95 upper .95
diabetes      2.31      0.432      1.29      4.16

Rsquare= 0.029   (max possible= 0.5 )
Likelihood ratio test= 8.55  on 1 df,   p=0.00345
Wald test            = 7.85  on 1 df,   p=0.00508
Score (logrank) test = 8.32  on 1 df,   p=0.00392
```

In Section 10.2.6, Agresti shows how to fit this model using only software for logistic regression.

We can also fit a generalized linear mixed model (GLMM), where the α_i are random effects with normal distribution, mean 0 and unknown standard deviation. The MASS library fits GLMMs with its function `glmmPQL` via penalized quasi-likelihood estimation. It uses the function `lme`, so the specification of its arguments is very similar to `lme`, and its output is identical. Two additional R packages have functions for fitting GLMMs. `glmmML` in package `glmmML` fits GLMMs (binomial and Poisson families only) with random (normal) intercepts via maximum likelihood estimation. The `repeated` package also contains a `glmm` function, which fits via ML estimation. Gauss-Hermite quadrature is used to integrate with respect to the random effects.

To use `glmmPQL`, we need to specify a fixed-effect formula and a random-effect formula. The fixed-effect formula will be like an ordinary S formula: response ~ predictor. Here, it is `MI ~ diabetes` because diabetes is the predictor. To specify a random intercept within pair, we use `~ 1 | pair` for the `random` argument. The `1` indicates a random intercept, and "`| pair`" means "within pair". Thus, the call for both S-PLUS and R is

```
library(MASS)
fit.glmmPQL<-glmmPQL(MI ~ diabetes, random = ~ 1 | pair, family = binomial, data =
    table.10.3)
summary(fit.glmmPQL)

iteration 1
Linear mixed-effects model fit by maximum likelihood
 Data: table.10.3
       AIC      BIC    logLik
  1232.324 1246.976 -612.1621

Random effects:
 Formula:  ~ 1 | pair
        (Intercept)  Residual
StdDev: 0.001827267 0.9993246

Variance function:
 Structure: fixed weights
 Formula:  ~ invwt
Fixed effects: MI ~ diabetes
                Value Std.Error  DF   t-value p-value
(Intercept) -0.1941562 0.1367889 143 -1.419386  0.1580
   diabetes  0.8038742 0.2836710 143  2.833826  0.0053

Number of Observations: 288
Number of Groups: 144
```

In the output, under “Random effects”, we get the standard deviation of the normal distribution of the random effects, which is quite small at `0.001827267`. This may imply a very low association between members of the same pair, as it means that the α_i are probably very similar (Recall that the association is described completely by the α_i, by definition of the model). “Residual”, I believe, is an estimate of the scale, which for the binomial is fixed at 1.0. We also get estimates of the fixed effects, notably the estimate of the effect of diabetes (`0.8038742`). The PQL estimate is slightly different from the conditional ML estimate from `coxph`.

The function `glmmML` only allows a random intercept, and as such is less general than `glmmPQL`, although it is fine for this particular problem. The arguments for `glmmML` differ in that `cluster` defines the random factor. Note that the `cluster` argument will be evaluated in the calling environment, not within the `data` list. So, you cannot just set `cluster=pair` for analyzing `table.10.3` above (you must include the data frame as well). Using `glmmML`, we get the same numerical estimate of β as from `glmmPQL`. The estimate of the standard deviation of the random effects (`7.12e-06`) is much smaller, but the resulting conclusion about low association is the same.

```
library(glmmML)
glmmML(MI ~ diabetes, cluster=table.10.3$pair, family = binomial, data = table.10.3)

               coef se(coef)      z Pr(>|z|)
(Intercept) -0.1942   0.1364 -1.423  0.15500
diabetes     0.8039   0.2835  2.836  0.00457

Standard deviation in mixing distribution:  7.12e-06
Std. Error:                                 0.05206

Residual deviance: 390.9 on 285 degrees of freedom      AIC: 396.9
```

The definitions of AIC are the same in both functions (i.e., `-2*loglik + 2*npar`). Thus, the difference in the AIC values across functions is due to the representation of the log likelihood, as `glmmML` gives `-195.4561` for the log likelihood.

The function `glmm` in the `repeated` library also fits random effects model, and allows several choices of error distributions besides binomial. The argument `nest` is for specifying the pair variable.

```
fit.glmm<-glmm(MI ~ diabetes, family=binomial, data=table.10.3, nest=pair)

summary(fit.glmm)

Coefficients:
             Estimate Std. Error  z value Pr(>|z|)
(Intercept) -1.942e-01  1.365e-01   -1.423   0.1548
diabetes     8.039e-01  2.837e-01    2.834   0.0046 **
sd           6.829e-07  1.196e-01 5.71e-06   1.0000
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 399.25  on 287  degrees of freedom
Residual deviance: 390.80  on 285  degrees of freedom
AIC: 396.8

Number of Fisher Scoring iterations: 4

Normal mixing variance: 4.662868e-13
```

Thus, `glmm` gives the same estimates as `glmmML`.

D. Marginal Models for Square Contingency Tables

This section discusses marginal models for matched-pairs with ordinal and nominal multinomial responses. For an I x I table, marginal homogeneity is $P(Y_1 = a) = P(Y_2 = a)$ for responses Y_t, t = 1, 2 and categories $a = 1,...,I$.

1. Ordinal Classifications

The cumulative logit model for matched-pairs is

$$\text{logit}\left[P(Y_t \le j)\right] = \alpha_j + \beta x_t, \quad t = 1,2; \quad j = 1,...,I-1$$

where $x_1 = 0$ and $x_2 = 1$. In this model, the odds of outcome $Y_2 \le j$ is $\exp(\beta)$ times the odds of outcome $Y_1 \le j$. Marginal homogeneity corresponds to $\beta = 0$.

In the Premarital and Extramarital Sex example, subjects responded with opinions about premarital sex (1 = always wrong, 2 = almost always wrong, 3 = sometimes wrong, 4 = never wrong) and about extra-marital sex. Agresti fits a cumulative marginal logit model (equation 10.14) to these data, using maximum likelihood estimation. One way to do this is to use J. Lang's `mph.fit` function for R, which uses a method of constrained maximum likelihood estimation. The model specification follows that in Section 11.2.5 in Agresti. Briefly, if $\boldsymbol{\pi}$ denotes the complete set of multinomial joint probabilities, then a marginal logit model has the generalized loglinear form

$$\mathbf{C}\log(\mathbf{A}\boldsymbol{\pi}) = \mathbf{X}\boldsymbol{\beta} \qquad (10.1)$$

The components of (10.1) are used in `mph.fit`.

In using the function, I follow the notation of Lang's documentation. First, I will fit a marginal homogeneity model, which constrains the corresponding row and column marginal probabilities to be equal. This constraint is specified in the function `h.fct`, which is an optional argument to `mph.fit`. As you might perceive, the h function is the gradient of the Lagrangian.

The observed counts are in `y`. The matrix Z describes the strata in the data. Here, we do not have stratification, so Z is a vector of ones. ZF = Z because the sample size is assumed fixed (see the documentation). `M1`, `M2`, and `C.matrix` are used in the constraint function. `M1` and `M2` are used to get row and column totals of the expected counts. `C.matrix` is a constraint matrix that specifies marginal homogeneity.

```
y <- c(144, 33, 84, 126, 2, 4, 14, 29, 0, 2, 6, 25, 0, 0, 1, 5)

ZF <- Z <- matrix(1,16,1)
```

The function `Marg.fct` creates a marginalizing matrix that when multiplied by the vector of counts gives the appropriate marginal counts. The first argument gives the index to NOT sum over. The second argument gives the levels of the factors (here, both have 4 levels).

```
M1 <- Marg.fct(1,rep(4,2))  # used to get m1+, etc
M2 <- Marg.fct(2,rep(4,2))  # used to get m+1, etc
```

The constraint matrix is used to specify marginal homogeneity, but doesn't use all equality relations, as some are redundant.

```
C.matrix <- matrix(c(
              1, 0, 0, 0, -1, 0, 0, 0,   # y1+ = y+1
              0, 1, 0, 0, 0, -1, 0, 0,   # y2+ = y+2
              0, 0, 1, 0, 0, 0, -1, 0),  # y3+ = y+3
```

```
      3,8,byrow=T)

h.fct <- function(m) {     # constraint function
   marg <- rbind(M1%*%m,  M2%*%m)    # y1+, y2+, y3+, y4+, y+1, y+2, y+3, y+4
   C.matrix%*%marg                   # y1+ = y+1, y2+ = y+2, etc
}
```

Now, we fit the model and get a summary, which gives the LR statistic, among other statistics. The p-values indicate a significantly poor fit.

```
a <- mph.fit(y=y,Z=Z,ZF=ZF,h.fct=h.fct)

mph.summary(a)

OVERALL GOODNESS OF FIT: TEST of   Ho: h(m)=0 vs. Ha: not Ho...
    Likelihood Ratio Stat (df= 3 ):  Gsq =  348.0921 (p =  0 )
    Pearson's Score Stat  (df= 3 ):  Xsq =  247.3116 (p =  0 )
    Generalized Wald Stat (df= 3 ):  Wsq =  271.8397 (p =  0 )


CONVERGENCE STATISTICS...
    iterations = 100
    norm.diff  = 0.368841
    norm.score = 7.2566e-09
    Original counts used.

FITTING PROGRAM USED:  mph.fit, version 1.0, 6/5/02
```

Now, to fit the cumulative marginal logit model, we need to create the C, A, and X matrices in equation (10.1). The matrices `y`, `Z`, `M1`, and `M2` remain the same as above. The result of the computations below give `AMarg`, which is a vector with the following sums of marginal probabilities, and their complements.

$$(\pi_{1+}, \pi_{2+}+\pi_{3+}+\pi_{4+}, \pi_{1+}+\pi_{2+}, \pi_{3+}+\pi_{4+}, \pi_{1+}+\pi_{2+}+\pi_{3+}, \pi_{4+}, \pi_{+1}, \pi_{+2}+\pi_{+3}+\pi_{+4}, ..., \pi_{+4})$$

```
M<-rbind(M1,M2)

CUM0 <- matrix(c(
             1, 0, 0, 0,
             0, 1, 1, 1,
             1, 1, 0, 0,
             0, 0, 1, 1,
             1, 1, 1, 0,
             0, 0, 0, 1)
      ,6,4,byrow=T)

CUM <- kronecker(diag(2),CUM0)

AMarg <- CUM%*%M
```

The C matrix (called `CMarg`) is a contrast matrix used to take differences of the logs of the probabilities above. When `CMarg` and `AMarg` are used in `L.fct`, which is the left side of equation (10.1), we get the cumulative logits.

```
C0 <- matrix(c(
             1, -1, 0, 0, 0, 0,
             0, 0, 1, -1, 0, 0,
             0, 0, 0, 0, 1, -1),
      3,6,byrow=T)
```

```
CMarg <- kronecker(diag(2),C0)


L.fct <- function(m) {
   CMarg%*%log(AMarg%*%m)
}
```

The design matrix on the right side of (10.1) represents the right side of model equation 10.14 in Agresti.

```
XMarg <- matrix(c(
             1, 0, 0, 1,   # j = 1, t = 1
             0, 1, 0, 1,   # j = 2, t = 1
             0, 0, 1, 1,   # j = 3, t = 1
             1, 0, 0, 0,   # j = 1, t = 0
             0, 1, 0, 0,   # etc
             0, 0, 1, 0)
      ,6, 4,byrow=T)
```

Now, we fit the model, and get a summary. I give selected output below.

```
a <- mph.fit(y,Z,ZF,L.fct=L.fct,X=XMarg)

mph.summary(a)

OVERALL GOODNESS OF FIT: TEST of   Ho: h(m)=0 vs. Ha: not Ho...
    Likelihood Ratio Stat (df= 2 ):  Gsq =  35.0301 (p =  2.4735e-08 )
    Pearson's Score Stat  (df= 2 ):  Xsq =  29.93261 (p =  3.1639e-07 )
    Generalized Wald Stat (df= 2 ):  Wsq =  24.56912 (p =  4.6226e-06 )


LINEAR PREDICTOR MODEL RESULTS...
            BETA StdErr(BETA)   Z-ratio      p-value
beta1 -0.9738808   0.09819651 -9.917673 0.000000e+00
beta2 -0.4330636   0.09185925 -4.714426 2.423929e-06
beta3  0.5385457   0.09246113  5.824564 5.726206e-09
beta4  2.4992700   0.12930736 19.328134 0.000000e+00



CONVERGENCE STATISTICS...
    iterations = 100
    norm.diff  = 1.24881
    norm.score = 0.0394783
    Original counts used.

FITTING PROGRAM USED:  mph.fit, version 1.0, 6/5/02
```

We get a much smaller LR statistic, but the model still fits significantly poorly. The term `beta4` above corresponds to β in Agresti's equation 10.14. The others correspond to the α_j's.

To use S-PLUS, you could either modify `mph.fit` to work with S-PLUS 6.1 or use the `yags` library, which is discussed in Chapter 11.

2. Nominal Classifications

For an I x I table with matched pairs, a LR test of marginal homogeneity maximizes a multinomial likelihood in the cell counts subject to corresponding marginal totals being equal. This can be done easily using `glm` in R or S-PLUS, or using `mph.fit` in R. To illustrate, we use the Migration Example in Section 10.3.4 in Agresti. The table of counts indicates the number of people surveyed who changed residence

from the Northeast, Midwest, South, and West regions of the U.S. in 1980 to any of those same regions in 1985. The large majority remained in the same region for both years.

To compute a LR test using `glm`, we need the cell counts and a set of dummy variables that correspond to the constraints of marginal homogeneity. Table A.17 in Agresti shows the set of dummy variables. In that table, corresponding marginal expected frequencies are constrained to be equal and are thus represented by only three variables: m1, m2, and m3 (m4 = n – m1 – m2 – m3), instead of six variables, differentiating row from column totals. Then, the observed counts are modeled as functions of the expected marginal frequencies as well as expected cell counts. For example, the observed (1, 4) cell is modeled as m1 – (m11 + m12 + m13), where mij gives the expected count in the ijth cell.

So, we create a data frame with this matrix of dummy variables and the observed counts.

```
dummies<-matrix(c(
 1,    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,    1,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
-1,   -1,   -1,    0,    0,    0,    0,    0,    0,    0,    1,    0,    0,
 0,    0,    0,    1,    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,    1,    0,    0,    0,    0,    0,    0,    0,
 0,    0,    0,   -1,   -1,   -1,    0,    0,    0,    0,    0,    1,    0,
 0,    0,    0,    0,    0,    0,    1,    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,    1,    0,    0,    0,    0,
 0,    0,    0,    0,    0,    0,   -1,   -1,   -1,    0,    0,    0,    1,
-1,    0,    0,   -1,    0,    0,   -1,    0,    0,    0,    1,    0,    0,
 0,   -1,    0,    0,   -1,    0,    0,   -1,    0,    0,    0,    1,    0,
 0,    0,   -1,    0,    0,   -1,    0,    0,   -1,    0,    0,    0,    1,
 0,    0,    0,    0,    0,    0,    0,    0,    0,    1,    0,    0,    0),
nrow=16, ncol=((4-1)^2) + 1 +3)


dummies<-data.frame(
   counts=c(11607,100,366,124,87,13677,515,302,172,225,17819,270,63,176,286,10192),
   dummies)
names(dummies)<-c("counts",
   "m11","m12","m13","m21","m22","m23","m31","m32","m33","m44","m1","m2","m3")
```

We then use `glm` with Poisson family and identity link to get the equivalencies corresponding to marginal homogeneity.

```
(fit<-glm(counts~.-1,family=poisson(identity),data=dummies))

Coefficients:
   m11      m12      m13      m21   m22      m23      m31      m32
 11607 98.08285 265.6867 88.73298 13677 379.0733 276.4748 350.8006

   m33   m44       m1       m2       m3
 17819 10192 12064.74 14377.15 18733.53

Degrees of Freedom: 16 Total; 3 Residual
Residual Deviance: 240.7458
```

The Residual Deviance gives the value of the LR statistic. Note that the estimated coefficients are the expected counts. We can also reproduce them using `fitted`.

```
residence80 <- (c("NE", "MW", "S", "W"))
residence85 <- (c("NE", "MW", "S", "W"))
matrix(fitted(fit), byrow = T, nc = 4, dimnames = list(residence80, residence85))

            NE          MW          S           W
```

```
NE 11607.00000    98.08285   265.6867    93.96975
MW    88.73298 13677.00000   379.0733   232.34523
 S   276.47479   350.80060 17819.0000   287.25936
 W    92.53157   251.26811   269.7747 10192.00000
```

Of course, with such a large value of the statistic, we reject marginal homogeneity.

The R function `mph.fit` can also be used to get the LR test. Example 3 of Lang's documentation for the function gives the necessary steps, which can be applied almost verbatim to the Migration data. It is left to the reader to try `mph.fit` on this data set, recalling that `h.fct` is the constraint function.

A Wald test of marginal homogeneity can be computed using Bhapkar's statistic (equation 10.16 in Agresti). This statistic is actually very easy to calculate if we follow Wickens (1989, pp. 313-314) and note that what we are testing is a set of simultaneous linear combinations of the cell probabilities. These linear combinations are

$$\pi_{12}+\pi_{13}+\pi_{14}=\pi_{21}+\pi_{31}+\pi_{41}$$
$$\pi_{12}+\pi_{32}+\pi_{42}=\pi_{21}+\pi_{23}+\pi_{24}$$
$$\pi_{13}+\pi_{23}+\pi_{43}=\pi_{31}+\pi_{32}+\pi_{34}$$

which imply marginal homogeneity if they hold simultaneously. If we represent the cell probabilities in a 16 x 1 vector, $\boldsymbol{\pi}$, then the following matrix, **A**, gives the three linear combinations above if post-multiplied by $\boldsymbol{\pi}$

```
A<-matrix(c(0,1,1,1,-1,0,0,0,-1,0,0,0,-1,0,0,0,
            0,1,0,0,-1,0,-1,-1,0,1,0,0,0,1,0,0,
            0,0,1,0,0,0,1,0,-1,-1,0,-1,0,0,1,0),nc=16,nr=3,byrow=T)
```

Post-multiplying **A** by the vector of sample proportions, **p**, gives the estimated linear combinations, **y**.

```
counts<-c(11607,100,366,124,87,13677,515,302,172,225,17819,270,63,176,286,10192)
p<-counts/(n<-sum(counts))
(y<-A%*%p)
```

```
              [,1]
[1,]  0.004787339
[2,] -0.007198871
[3,]  0.008931602
```

To compute estimated covariance matrix of **y**, we need is the estimated covariance matrix of **p**. This is found easily using the `outer` function applied to **p**, and then setting the diagonal.

```
Sp<--outer(p,p)/n
diag(Sp)<-p*(1-p)/n
```

Then, **Sy** is the estimated covariance matrix of **y**.

```
Sy<-A%*%Sp%*%t(A)
```

The Wald statistic is the quadratic form in **y**, below.

```
W<-as.numeric(t(y) %*% solve(Sy) %*% y)
```

```
[1] 236.4906
```

which compared to a chi-squared distribution with 3 df gives a p-value of

```
1 - pchisq(W, df = 3)
```

```
[1] 0
```

E. Symmetry, Quasi-symmetry, and Quasi-independence

An I x I table satisfying symmetry has $\pi_{ab} = \pi_{ba}$, $a \neq b$. This implies the loglinear model

$$\log \mu_{ab} = \lambda + \lambda_a + \lambda_b + \lambda_{ab}$$

where all $\lambda_{ab} = \lambda_{ba}$. Note that the main-effect terms are the same for the two expected frequencies μ_{ab} and μ_{ba}. Quasi-symmetry allows the main-effect terms to differ so that

$$\log \mu_{ab} = \lambda + \lambda_a^X + \lambda_b^Y + \lambda_{ab} \neq \log \mu_{ba} = \lambda + \lambda_b^X + \lambda_a^Y + \lambda_{ab}$$

For this model, the odds ratios on one side of the main diagonal are identical to the "mirror-image" odds ratios on the other side. The loglinear model for a table displaying quasi-independence adds a parameter for each cell on the main diagonal:

$$\log \mu_{ab} = \lambda + \lambda_a^X + \lambda_b^Y + \delta_a I(a = b)$$

When $\delta_a > 0$, μ_{aa} is greater than under independence. Thus, this model would be used for a table that showed independence on the off-diagonal cells, but had larger counts on the main diagonal.

Agresti uses the Migration example to illustrate fitting these three models via maximum likelihood estimation. We can use the definitions above to create a data frame

```
residence80<-factor(residence80, levels= residence80)
residence85<-residence80

table.10.6<-expand.grid(res80=residence80,res85=residence85)
table.10.6$counts<-c(11607,100,366,124,87,13677,515,302,172,225,17819,270,
   63,176,286,10192)
```

1. Symmetry Model

To fit a symmetry model, we add a factor called "symm" to the data frame. This factor has a level for each unique cell probability in the table. I also reverse the levels so that the coefficient that is zeroed out for identifiability is the last coefficient.

```
table.10.6$symm<-paste(
   pmin(as.numeric(table.10.6$res80),as.numeric(table.10.6$res85)),
   pmax(as.numeric(table.10.6$res80),as.numeric(table.10.6$res85)),sep=",")
table.10.6$symm<-factor(table.10.6$symm, levels=rev(table.10.6$symm))
```

```
   res80 res85 counts symm
1     NE    NE  11607  1,1
2     MW    NE    100  1,2
3      S    NE    366  1,3
4      W    NE    124  1,4
5     NE    MW     87  1,2
6     MW    MW  13677  2,2
7      S    MW    515  2,3
8      W    MW    302  2,4
9     NE     S    172  1,3
```

```
10     MW      S     225  2,3
11      S      S   17819  3,3
12      W      S     270  3,4
13     NE      W      63  1,4
14     MW      W     176  2,4
15      S      W     286  3,4
16      W      W   10192  4,4
```

```
options(contrasts = c("contr.treatment", "contr.poly"))
(fit.symm <- glm(counts ~ symm, family = poisson(log), data = table.10.6))
```

```
Coefficients:
 (Intercept)    symm3,4    symm2,4    symm1,4    symm3,3    symm2,3    symm1,3    symm2,2
    9.229358  -3.601737  -3.752895  -4.691396  0.5586622  -3.315851  -3.634645  0.2941125

   symm1,2    symm1,1
 -4.691397  0.1300053

Degrees of Freedom: 16 Total; 6 Residual
Residual Deviance: 243.5502
```

This method is due to Alan Zaslavsky, as far as I know. The estimates in the model correspond to the λ_{ab} s in the model $\log \mu_{ab} = \lambda_{ab}$. The deviance statistic is still very high..

2. Quasi-Symmetry Model

For the quasi-symmetry, we need to add a factor to the model that differentiates main effects for rows and columns. We do this by adding `res80a` to the model, which is res80 with the levels reversed so that the last category coefficient (for West) is zeroed out. The resulting coefficient estimates are the differences $\left\{\lambda_j^Y - \lambda_j^X\right\}$ for j = 1, 2, 3.

```
options(contrasts = c("contr.treatment", "contr.poly"))
table.10.6$res80a<-factor(table.10.6$res80, levels=rev(residence80))
table.10.6$res85a<-factor(table.10.6$res85, levels=rev(residence80))  # we will use
   this for quasi-independence
(fit.qsymm<-glm(counts~symm+res80a,family=poisson(log),data=table.10.6))
```

```
Coefficients:
 (Intercept)    symm3,4    symm2,4    symm1,4    symm3,3    symm2,3    symm1,3  symm2,2
    9.229358  -3.664378  -3.489224  -4.410902  0.4370738  -3.132979  -3.436262 0.916888

   symm1,2    symm1,1    res80aS   res80aMW   res80aNE
 -4.044439  0.8017449  0.1215884 -0.6227755 -0.6717397

Degrees of Freedom: 16 Total; 3 Residual
Residual Deviance: 2.985962
```

The quasi-symmetry model fits well, with asymptotic p-value

```
1 - pchisq(fit.qsymm$deviance, df = fit.qsymm$df)
[1] 0.3937946
```

which well exceeds 0.05.

To interpret the parameter estimates in terms of subject-specific effects, for a given subject, the estimated odds of living in the MidWest instead of the West (the last category) in 1985 were exp(-0.623) = 0.536 times the odds in 1980.

Fitted counts are obtained using

```
matrix(fitted(fit.qsymm), ncol = 4, byrow = T, dimnames =
   list(rev(levels(residence80)),  rev(levels(residence85))))
```

```
            NE          MW          S          W
NE 11607.00000    95.78862   370.4375   123.7739
MW    91.21138 13677.00000   501.6825   311.1061
 S   167.56253   238.31746 17819.0000   261.1200
 W    63.22609   166.89392   294.8800 10192.0000
```

From equation (10.25) in Agresti, a test of the null hypothesis of marginal homogeneity is given by

```
1 - pchisq(fit.symm$deviance - fit.qsymm$deviance, df = fit.symm$df - fit.qsymm$df)
```

```
[1] 0
```

showing strong evidence of marginal heterogeneity.

A fit of the quasi-symmetry model for the Migration data is given in an example in the documentation for the R package `exactLoglinTest`. In fact, the data are included in the library as `residence.dat`.

```
library(exactLoglinTest)
data(residence.dat)
```

```
residence.dat
```

```
       y res.1985 res.1980 sym.pair
1  11607       NE       NE        1
2    100       MW       NE        2
3    366        S       NE        3
4    124        W       NE        4
5     87       NE       MW        2
6  13677       MW       MW        5
7    515        S       MW        6
8    302        W       MW        7
9    172       NE        S        3
10   225       MW        S        6
11 17819        S        S        8
12   270        W        S        9
13    63       NE        W        4
14   176       MW        W        7
15   286        S        W        9
16 10192        W        W       10
```

The variable `sym.pair` serves the same purpose as `symm` above. We can use `mcexact` to check the asymptotic p-value.

```
(fit.qsymm.exact<-mcexact(y ~ res.1980 + factor(sym.pair), data=residence.dat,
maxiter=10^6,  nosim=10^4))
```

```
                deviance   Pearson
observed.stat 2.985962330 2.9819870
pvalue        0.395651242 0.3956424
mcse          0.003848147 0.0038481
```

The Monte Carlo p-value is slightly larger.

3. Quasi-independence

To fit a quasi-independence model to the Migration data, we can add four dummy variables to the data frame that represent the main diagonal cells.

```
table.10.6$D1<-as.numeric(table.10.6$symm=="1,1")
table.10.6$D2<-as.numeric(table.10.6$symm=="2,2")
table.10.6$D3<-as.numeric(table.10.6$symm=="3,3")
table.10.6$D4<-as.numeric(table.10.6$symm=="4,4")
```

The model is fit with these variables plus the main effect variables

```
options(contrasts = c("contr.treatment", "contr.poly"))
 (fit.qi <- glm(counts ~ res80a + res85a + D1 + D2 + D3 + D4, family = poisson(log),
   data = table.10.6))

Coefficients:
 (Intercept)   res80aS   res80aMW   res80aNE   res85aS  res85aMW   res85aNE       D1
    5.045833 0.7317538 -0.1734236 -0.7777343 0.6239489 0.4973868 -0.0316759 5.122941

       D2       D3       D4
 4.153674 3.386485 4.183525


Degrees of Freedom: 16 Total; 5 Residual
Residual Deviance: 69.5094
```

This model fits worse than the quasi-symmetry model.

The first edition of this manual (Thompson, 1999) gave another way to fit the quasi-independence model using `loglin` (or `loglm` in MASS). With this method, one zeroes out the main diagonal of the table and fits an independence loglinear model, treating the main diagonal zeroes as structural zeroes. The `start` option in `loglin` can be used to set structural zeroes. Note that this method only deals with the part of the table relevant for the analysis (i.e., the off-diagonals).

F. Square Tables with Ordered Categories

When categories are ordered, more parsimonious loglinear models exist. Agresti discusses quasi-symmetry models, conditional symmetry models, and quasi-uniform association models for ordinal categories.

1. Ordinal Quasi-Symmetry

For ordered scores $u_1 \leq ... \leq u_I$, ordinal quasi-symmetry model sets $\lambda_b^Y - \lambda_b^X = \beta u_b$ within the quasi-symmetry model for nominal categories. Thus, the difference in the effect of a category from one occasion to the other follows a linear trend in the category scores. The logit representation (10.27 in Agresti) shows that the greater $|\beta|$, the greater the difference between the two joint probabilities π_{ab} and π_{ba} and hence the difference between the marginal row and column distributions. Thus, a test of marginal homogeneity is a test of $\beta = 0$. As with quasi-symmetry models for nominal categories, a LR test compares the deviance for symmetry and quasi-symmetry.

Agresti fits an ordinal quasi-symmetry model to the Premarital and Extra-marital Sex data, giving scores 1, 2, 3, 4 to the categories. We can use `glm` to fit this model.

Again, when we create a `symm` factor, we use the reverse levels so that the 4,4 coefficient is zeroed out.

```
table.10.5<-data.frame(expand.grid(PreSex=factor(1:4),ExSex=factor(1:4)),counts=c(144,
   33, 84, 126, 2, 4 ,14 ,29 ,0 ,2 ,6 ,25, 0, 0, 1, 5))
```

```
table.10.5$symm<-paste(
   pmin(as.numeric(table.10.5$PreSex),as.numeric(table.10.5$ExSex)),
   pmax(as.numeric(table.10.5$PreSex),as.numeric(table.10.5$ExSex)),sep=",")
table.10.5$symm<-factor(table.10.5$symm, levels=rev(table.10.5$symm))

table.10.5$scores<-rep(1:4,each=4)

options(contrasts = c("contr.treatment", "contr.poly"))
(fit.oqsymm<-glm(counts~symm + scores,data=table.10.5,family=poisson(log))   )

Coefficients:
 (Intercept)   symm3,4   symm2,4   symm1,4   symm3,3   symm2,3  symm1,3   symm2,2
    13.03569 -1.263781 -3.958567 -5.343038 -2.674243 -4.605853 -5.75161 -5.936272

   symm1,2   symm1,1    scores
 -6.679657 -5.209317 -2.856564

Degrees of Freedom: 16 Total; 5 Residual
Residual Deviance: 2.097209
```

Thus, the estimated probability that premarital sex is judged wrong only sometimes and extramarital sex is judged always wrong is exp(2*2.86) = 304.9 times the estimated probability that extramarital sex is judged wrong only sometimes and premarital sex is judged always wrong.

2. Conditional Symmetry

Symmmetry in an I x I table implies $\pi_{ab} = \pi_{ba}$ for all a, b. Conditional symmetry implies $\pi_{ab} < \pi_{ba}$ or $\pi_{ab} > \pi_{ba}$ for all a < b. Thus, $\pi_{ab} = e^{\tau}\pi_{ba}$ for some τ, which implies that the probability of first response being a and second response being b is e^{τ} larger than the probability of first response a and second response b.

This gives the logit model

$$\log(\pi_{ab}/\pi_{ba}) = \tau, \qquad a < b$$

The equivalent loglinear model is

$$\log \mu_{ab} = \lambda + \lambda_a + \lambda_b + \lambda_{ab} + \tau I(a < b)$$

where all $\lambda_{ab} = \lambda_{ba}$.

To fit the conditional symmetry model to the example above, first I get a vector that represents τ.

```
temp<-matrix(0,nr=4,nc=4)
tau<-as.numeric(row(temp)<col(temp))
tau

 [1] 0 0 0 0 1 0 0 0 1 1 0 0 1 1 1 0
```

Then, I add tau to the symmetry model

```
options(contrasts = c("contr.treatment", "contr.poly"))
fit.cs<-glm(counts~symm+tau,family=poisson(log),data=table.10.5)
summary(fit.cs, cor=F)

Coefficients:
```

```
                Value Std. Error    t value
(Intercept)  1.6094379  0.4472136  3.5988126
    symm3,4  1.6327093  0.4883751  3.3431460
    symm2,4  1.7419086  0.4842847  3.5968686
    symm1,4  3.2108947  0.4560560  7.0405708
    symm3,3  0.1823216  0.6055301  0.3010941
    symm2,3  1.1472015  0.5123936  2.2389066
    symm1,3  2.8054296  0.4603858  6.0936490
    symm2,2 -0.2231436  0.6708204 -0.3326428
    symm1,2  1.9299608  0.4781430  4.0363673
    symm1,1  3.3603754  0.4549115  7.3868777
        tau -4.1303475  0.4499372 -9.1798320

(Dispersion Parameter for Poisson family taken to be 1 )

    Null Deviance: 888.3778 on 15 degrees of freedom

Residual Deviance: 15.51736 on 5 degrees of freedom

Number of Fisher Scoring Iterations: 5
```

The estimated value of τ implies that the estimated probability that premarital sex is considered more wrong (i.e., closer to 1 than 4) than extramarital sex is exp(-4.13) = 0.016 times the estimated probability that extramarital sex is considered more wrong.

3. Quasi-Uniform Association

Even after conditioning on the event that the two responses differ, sometimes independence still does not hold. Sometimes there is a monotone pattern to the probabilities. The loglinear model

$$\log \mu_{ab} = \lambda + \lambda_a^X + \lambda_b^Y + \beta u_a u_b + \delta_a I(a = b)$$

permits linear-by-linear association off the main diagonal. For equal-interval scores, it implies *uniform* local association, given that responses differ.

Fitting this model to the above example requires first defining scores for the row variable, then creating a Delta vector respresenting δ_a.

```
table.10.5$scores.a<-rep(1:4,4)
Delta<-as.numeric(row(temp)==col(temp))
```

I also modified the PreSex and ExSex factors to have reversed levels so that the last category coefficient would be zero.

```
table.10.5$PreSex<-factor(table.10.5$PreSex, levels=rev(table.10.5$PreSex))
table.10.5$ExSex<-factor(table.10.5$ExSex, levels=rev(table.10.5$ExSex))

options(contrasts = c("contr.treatment", "contr.poly"))
fit.qa<-glm(counts~PreSex+ExSex+scores:scores.a + Delta,family=poisson(log),
   data=table.10.5)
summary(fit.qa, cor=F)

Coefficients:
                     Value Std. Error   t value
    (Intercept)  -8.58112859  1.8763462 -4.573318
        PreSex3   0.24220678  0.1989176  1.217624
        PreSex2  -0.04925675  0.3280743 -0.150139
        PreSex1   1.54119021  0.4710163  3.272053
         ExSex3   4.37970228  0.6979412  6.275174
         ExSex2   7.06457536  1.0791369  6.546505
         ExSex1  10.94128024  1.4192651  7.709117
```

```
          Delta   0.43837201  0.2737843  1.601158
scores:scores.a   0.61632154  0.1142078  5.396493

(Dispersion Parameter for Poisson family taken to be 1 )

    Null Deviance: 888.3778 on 15 degrees of freedom

Residual Deviance: 5.797136 on 7 degrees of freedom

Number of Fisher Scoring Iterations: 4
```

Thus, off the main diagonal, the estimated local odds ratio is exp(0.616) = 1.85, which is slightly lower than Agresti's result of 1.88.

G. Measuring Agreement Between Observers

Matched-pairs models are used to measure agreement between ratings from two observers. An independence model would imply no agreement, whereas a quasi-independence or quasi-symmetry model implies some agreement in ratings.

Section 10.5 in Agresti uses the Pathologists' Tumor Ratings data to illustrate fitting different models to assess degree of agreement between the two pathologists. The baseline model is the independence model. The quasi-independence and quasi-symmetry models allow estimation of the odds of agreement.

The data set is available in the R package exactLoglinTest. We dump the object and source it into an S-PLUS session. So, for example

```
# R
library(exactLoglinTest)
data(pathologist.dat)

dump("pathologist.dat", file="c:/program files/insightful/splus61/users/cda/
pathologist.R")

# S-PLUS
source("pathologist.R")
```

However, there are three differences between pathologist.dat and Table 10.8 in Agresti. First, Agresti only uses the first 4 levels of each variable. Second, the A and B labels are reversed. Third, two of the counts differ across the two data sets. We will fix the last two now, then subset the data in the `glm` call.

We can switch the labels by changing the names

```
names(pathologist.dat)<-c("y","B","A")
```

We fix the two counts to match Agresti's

```
pathologist.dat$y[pathologist.dat$A==4 & (pathologist.dat$B==3 |
   pathologist.dat$B==4)] <-c(17,10)
```

We also create factors out of A and B, making the levels reversed so that the last category is zero for identifiability.

```
pathologist.dat$A<-factor(pathologist.dat$A, levels=5:1)
pathologist.dat$B<-factor(pathologist.dat$B, levels=5:1)
```

Agresti shows that the independence model fits poorly, with large positive standardized residuals on the main diagonal and mainly negative residuals on the off-diagonals.

```
options(contrasts = c("contr.treatment", "contr.poly"))
(fit.ind <- glm(y ~ A + B, data = pathologist.dat, family = poisson(log), subset = (A
   != 5) & (B != 5)))

Coefficients:
 (Intercept)        A3          A2          A1       B3        B2        B1
   0.8641481 0.3053828 -0.07410805 -0.07410781 1.931478 0.1822781 0.9932105

Degrees of Freedom: 16 Total; 9 Residual
Residual Deviance: 117.9569


pear.std<-resid(fit.ind, type="pearson")/sqrt(1-lm.influence(fit.ind)$hat)
matrix(pear.std,nr=4,byrow=T)

           [,1]       [,2]       [,3]       [,4]
[1,]  8.4792195 -0.473263 -5.9460585 -1.749755
[2,] -0.5014518  3.201010 -0.5419787 -1.749897
[3,] -4.0736603 -1.214994  5.5035585 -2.268019
[4,] -3.2976055 -1.322654  0.2751683  5.901283
```

A quasi-independence model would account for the counts on the main diagonal being larger than expected under independence. Recall this model adds main-diagonal parameters. We can add these parameters into the model by creating dummy variables

```
pathologist.dat$D1<-as.numeric((pathologist.dat$A==1) & (pathologist.dat$B==1))
pathologist.dat$D2<-as.numeric((pathologist.dat$A==2) & (pathologist.dat$B==2))
pathologist.dat$D3<-as.numeric((pathologist.dat$A==3) & (pathologist.dat$B==3))
pathologist.dat$D4<-as.numeric((pathologist.dat$A==4) & (pathologist.dat$B==4))

options(contrasts = c("contr.treatment", "contr.poly"))
(fit.qi<-glm(y~A+B+D1+D2+D3+D4,family=poisson(log),data=pathologist.dat,
   subset=(A!=5)&(B!=5)))

Coefficients:
 (Intercept)         A3         A2        A1        B3        B2        B1        D1
   -9.146122 -0.8928303 0.2375028 -1.39455 11.71995 10.25029 9.770648 3.861066

       D2       D3       D4
 0.6042435 1.902521 11.44871

Degrees of Freedom: 16 Total; 5 Residual
Residual Deviance: 13.17847
```

If two pathologists classify two slides as (2 = atypical squamous hyperplasia) and (3 = carcinoma in situ), then the odds that they agree on which is 2 and which is 3 are equal to

$$\tau_{23} = \frac{\pi_{22}\pi_{33}}{\pi_{23}\pi_{32}} = \frac{\mu_{22}\mu_{33}}{\mu_{23}\mu_{32}} = \exp(\delta_2 + \delta_3)$$

This is estimated as exp(0.6 + 1.9) = 12.3.

This model fits better than independence, but the fitted counts show some discrepancy compared with the observed counts.

```
matrix(fitted(fit.qi), nr = 4, byrow = T)

           [,1]       [,2]       [,3]           [,4]
[1,] 22.0000000 0.7479747  3.252025 2.643897e-005
[2,]  2.3679620 7.0000000 16.632038 1.352185e-004
```

```
[3,]  0.7646757 1.2353243 36.000000 4.366551e-005
[4,]  1.8673623 3.0167010 13.115937 1.000000e+001
```

A quasi-symmetry model would account for any association on the off-diagonal. To fit that model, we need to create a symm factor. To ensure that we use the correct levels, we create a new data frame without the ratings of 5. Then, we drop the unused level 5 in the factors.

```
pathologist.dat2<-pathologist.dat[(pathologist.dat$A!=5)&(pathologist.dat$B!=5),]
pathologist.dat2$A<-pathologist.dat2$A[drop=T]
pathologist.dat2$B<-pathologist.dat2$B[drop=T]
```

Because we already reversed the levels of A and B, the symm factor levels are not reversed.

```
pathologist.dat2$symm<-paste(
   pmin(as.numeric(pathologist.dat2$A),as.numeric(pathologist.dat2$B)),
   pmax(as.numeric(pathologist.dat2$A),as.numeric(pathologist.dat2$B)),sep=",")

pathologist.dat2$symm<-factor(pathologist.dat2$symm, levels=(pathologist.dat2$symm))
```

Because we have two symmetric cells both with a count of 0, we have to fix the fitted counts to zero. The function `loglin` can do this well through the use of the `start` argument (see p. 71 of Thompson, 1999). As we have been using `glm` here, we will try to achieve the same thing by using `start` (in S-PLUS) or `etastart` (in R).

```
starter<-fitted(fit.qi)
starter[c(4,13)]<-0     # the fixed zeroes

options(contrasts = c("contr.treatment", "contr.poly"))
fit.qsymm<-glm(y~symm + A, data=pathologist.dat2, family=poisson(log),
   control=glm.control(maxit=100), start=starter)
# R: fit.qsymm<-glm(y~symm + A, data=pathologist.dat2, family=poisson(log),
   control=glm.control(maxit=100), etastart=starter)

Coefficients:
 (Intercept)   symm3,4   symm2,4   symm1,4   symm3,3   symm2,3   symm1,3  symm2,2
    27.39476 -2.230336 -2.599296 -60.19605 -1.818099 -1.099226 -27.39476 1.992546

   symm1,2   symm1,1        A3        A2        A1
 -24.56154 -25.09217 -25.80378 -23.63075 -24.30371

Degrees of Freedom: 16 Total; 3 Residual
Residual Deviance: 0.9783039
```

The degrees of freedom apparently need to be reduced because we fixed cells. Under quasi-symmetry, if two pathologists classify two slides as (2 = atypical squamous hyperplasia) and (3 = carcinoma in situ), then the odds that they agree on which is 2 and which is 3 are equal to

```
as.numeric(exp(coefs["symm2,2"] + coefs["symm3,3"] - 2 * coefs["symm2,3"]))

[1] 10.72846
```

The fitted counts then should reflect the fixed zeroes.

```
matrix(fitted(fit.qsymm), nr = 4, byrow = T)

              [,1]     [,2]     [,3]          [,4]
[1,] 2.200000e+001 2.36483  1.63517 2.220446e-016
[2,] 4.635170e+000 7.00000 14.36483 5.461304e-011
[3,] 3.648303e-001 1.63517 36.00000 1.056836e-010
[4,] 5.683035e-015 1.00000 17.00000 1.000000e+001
```

H. Kappa Measure of Agreement

A numerical index that summarizes agreement is Cohen's Kappa. Kappa compares the probability of agreement, $\sum_a \pi_{aa}$, to that expected if the two sets of ratings were independent, $\sum_a \pi_{a+}\pi_{+a}$, as a proportion of the difference between perfect agreement (1.0) and chance agreement.

$$\kappa = \frac{\sum_a \pi_{aa} - \sum_a \pi_{a+}\pi_{+a}}{1 - \sum_a \pi_{a+}\pi_{+a}}$$

Kappa equals 0 when agreement equals that expected under independence, and equals 1.0 when perfect agreement occurs. With ordered categories, one can weight disagreements in ratings differently depending on the difference between the ratings. Weighted Kappa is

$$\kappa_w = \frac{\sum_a \sum_b w_{ab}\pi_{ab} - \sum_a \sum_b w_{ab}\pi_{a+}\pi_{+b}}{1 - \sum_a \sum_b w_{ab}\pi_{a+}\pi_{+b}}$$

Fleiss and Cohen give weights $w_{ab} = 1 - (a-b)^2 / (I-1)^2$, which are larger for ratings closer together.

We can compute Kappa and its ASE in S-PLUS using the function

```
Kappa<-function(table){
   pow<-function(x,a) x^a
   pij<-table/(n<-sum(table))
   row.sums<-rowSums(pij)
   col.sums<-colSums(pij)
   outer.sum<-outer(row.sums,col.sums,"+")
   pio<-sum(diag.pij<-diag(pij))
   pie<-sum(row.sums*col.sums)

   kap<-(pio-pie)/(1-pie)

   var<-(pio*(1-pio)/pow(1-pie,2)) + (2*(1-pio)*(2*pio*pie-sum(diag.pij*(row.sums +
   col.sums)))/pow(1-pie,3)) +
   pow(1-pio, 2)*(sum(pij*pow(outer.sum,2)) - 4*pow(pie,2))/pow(1-pie, 4)

   return(c(kappa=kap, SE=sqrt(var/n)))
}
```

which requires a table as input (obtainable using `design.table`, for example). In the R package `vcd`, there is a `Kappa` function which computes both unweighted and weighted Kappa. It also gives the estimated standard error and a 95% confidence interval. Thus, for the pathologists' tumor ratings data, we get

```
# S-PLUS
path.tab<-design.table(pathologist.dat2[c("y","B","A")])
Kappa(path.tab)

     kappa         SE
 0.4930056 0.06137286
```

and

```
# R
library(vcd)
```

```
path.tab<-xtabs(y~A+B, data=pathologist.dat2)
Kappa(path.tab, weights="Fleiss-Cohen")

              value        ASE       lwr       upr
Unweighted 0.4930056 0.06163942 0.3721945 0.6138166
Weighted   0.7838219 0.10260338 0.5827230 0.9849208
```

The 95% confidence interval shows that the weighted Kappa can be quite high. However, we don't have ordinal ratings. Some nominal classification disagreements may be considered more severe than others (e.g., one classification of 1 and another not 1) in which case the weights matrix would reflect that.

I. Bradley-Terry Model for Paired Preferences

Consider an I x I table of counts where the (*a*, *b*) cell is the number of times category *a* is preferred to category *b*, of a set of I categories (the main diagonal is empty). The Bradley-Terry model for the probability that *a* is preferred to *b* is

$$\log \frac{\Pi_{ab}}{\Pi_{ba}} = \beta_a - \beta_b$$

where Π_{ab} is the probability that *a* is preferred to *b*, and $\Pi_{ab} = \exp(\beta_a)/[\exp(\beta_a) + \exp(\beta_b)]$. There are $\binom{I}{2}$ such probabilities for the I x I table, described by $(I-1)$ parameters. Thus, the above Bradley-Terry model has residual df = $\binom{I}{2} - (I-1)$. When the N_{ab} comparisons of categories *a* and *b* are independent with probability Π_{ab} for each, then the number preferring *a*, n_{ab}, has a binomial(N_{ab}, Π_{ab}) distribution.

Agresti fits the Bradley-Terry model to wins (and losses) from the seven teams of the 1987 American baseball season in the Eastern Division. We want to estimate the probability that each team is preferred over each other team. We can do this by fitting the logit model as above or fitting the equivalent quasi-symmetry model in Section 10.6.3 in Agresti.

For the logit model formulation, we first create seven artificial variables, corresponding to the seven teams, and then fitting a logit model to the counts of wins. Here, X1 – X7 are the artificial variables, which equal 1 if the particular team won the game represented by the cell, – 1 if the team lost the game, and 0 if the team did not play.

```
Milwaukee<-c(-1,-1,-1,-1,-1,-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
Detroit<-  c(1,0,0,0,0,0,-1,-1,-1,-1,-1,rep(0,10))
Toronto<-  c(0,1,0,0,0,0,1,0,0,0,0,-1,-1,-1,-1,rep(0,6))
NY<-       c(0,0,1,0,0,0,0,1,0,0,0,1,0,0,0,-1,-1,-1,rep(0,3))
Boston<-   c(0,0,0,1,0,0,0,0,1,0,0,0,1,0,0,1,0,0,-1,-1,0)
Cleveland<-c(0,0,0,0,1,0,0,0,0,0,1,0,0,0,1,0,0,1,0,1,0,-1)
Baltimore<-c(0,0,0,0,0,1,0,0,0,0,1,0,0,0,1,0,0,1,0,1,0)
```

Then, the first column of `response` is the number of wins in each of the 21 games; the second is the number of losses.

```
response<-cbind(c(6,4,6,6,4,2,6,8,2,4,4,6,6,5,1,7,6,3,6,1,7), 13 - c(6,4,6,6,4,2,6,
8,2,4,4,6,6,5,1,7,6,3,6,1,7))
```

To fit the model, we use the assumption of the binomial distributions given above. Each of the Xs will represent a parameter. We set the last parameter to zero for identifiability purposes. We also exclude an intercept.

```
options(contrasts=c("contr.treatment", "contr.poly"))
fit.BT<-glm(response~-1 + Milwaukee + Detroit + Toronto + NY + Boston + Cleveland ,
    family=binomial)  # exclude intercept

summary(fit.BT, cor = F)

Coefficients:
             Value Std. Error  t value
Milwaukee 1.5813542  0.3430970 4.609059
  Detroit 1.4364067  0.3394029 4.232157
  Toronto 1.2944835  0.3365135 3.846750
       NY 1.2476162  0.3357076 3.716377
   Boston 1.1076961  0.3337292 3.319147
Cleveland 0.6838513  0.3317390 2.061414

(Dispersion Parameter for Binomial family taken to be 1 )

    Null Deviance: 49.69851 on 21 degrees of freedom

Residual Deviance: 15.7365 on 15 degrees of freedom

Number of Fisher Scoring Iterations: 3
```

The fitted counts are given by

```
losing.team<-c("Milwaukee","Detroit","Toronto","NY","Boston","Cleveland", "Baltimore")
win.team<-losing.team

fitted.counts<-matrix(0,nc=7,nr=7,dimnames=list(win.team,win.team))
fitted.counts[lower.tri(fitted.counts)]<-round(13*fitted(fit.BT),1)
fitted.counts[!lower.tri(fitted.counts,diag=T)]<-13-round(13*fitted(fit.BT),1)
fitted.counts
```

	Milwaukee	Detroit	Toronto	NY	Boston	Cleveland	Baltimore
Milwaukee	0.0	7.0	7.4	8.0	7.0	10.5	7.0
Detroit	6.0	0.0	7.6	9.2	7.1	6.7	8.3
Toronto	5.6	6.0	0.0	10.8	7.6	7.1	10.1
NY	5.4	5.9	6.3	0.0	8.8	8.4	7.9
Boston	5.0	5.4	5.9	6.0	0.0	10.2	9.8
Cleveland	3.8	4.2	4.6	4.7	5.1	0.0	8.6
Baltimore	2.2	2.5	2.8	2.9	3.2	4.4	0.0

The estimated probabilities of each team beating another are given by

```
fitted.probs<-matrix(0,nc=7,nr=7,dimnames=list(win.team,win.team))
fitted.probs[lower.tri(fitted.probs)]<-round(fitted(fit.BT),2)
fitted.probs[!lower.tri(fitted.probs,diag=T)]<-1-round(fitted(fit.BT),2)
```

	Milwaukee	Detroit	Toronto	NY	Boston	Cleveland	Baltimore
Milwaukee	0.00	0.54	0.57	0.62	0.54	0.81	0.53
Detroit	0.46	0.00	0.58	0.71	0.55	0.51	0.64
Toronto	0.43	0.46	0.00	0.83	0.58	0.55	0.78
NY	0.42	0.45	0.49	0.00	0.68	0.65	0.60
Boston	0.38	0.42	0.45	0.47	0.00	0.78	0.75
Cleveland	0.29	0.32	0.35	0.36	0.40	0.00	0.66
Baltimore	0.17	0.19	0.22	0.22	0.25	0.34	0.00

We can also fit the Bradley-Terry model as a quasi-symmetry model. Under that model, given that an observation is in either cell (*a*, *b*) or cell (*b*, *a*), the logit of the conditional probability that it falls in (*a*, *b*) is

$$\log\frac{\mu_{ab}}{\mu_{ba}} = (\lambda + \lambda_a^X + \lambda_b^Y + \lambda_{ab}) - (\lambda + \lambda_b^X + \lambda_a^Y + \lambda_{ab})$$
$$= (\lambda_a^X - \lambda_a^Y) - (\lambda_b^X - \lambda_b^Y) = \beta_a - \beta_b$$

We fit this quasi-symmetry model using the method from Section E of this chapter. First, I set up the data set. The `symm` factor represents the interaction term.

```
losing.team<-c("Milwaukee","Detroit","Toronto","NY","Boston","Cleveland","Baltimore")
win.team<-losing.team      # create labels

table.10.10<-expand.grid(losing=factor(losing.team,
   levels=rev(losing.team)),winning=factor(win.team, levels=rev(win.team)))
table.10.10$counts<-
   c(0,7,9,7,7,9,11,6,0,7,5,11,9,9,4,6,0,7,7,8,12,6,8,6,0,6,7,10,6,2,6,7,0,7,12,4,4,5,
   6,6,0,6,2,4,1,3,1,7,0)
table.10.10$symm<-paste(
   pmin(as.character(table.10.10$winning),as.character(table.10.10$losing)),
   pmax(as.character(table.10.10$winning),as.character(table.10.10$losing)),sep=",")
```

Now, I fit the model

```
options(contrasts=c("contr.treatment", "contr.poly"))
fit.BTQS<-glm(counts~symm+winning, data=table.10.10, family=poisson(log))
summary(fit.BTQS, cor = F)

Coefficients:
                          Value Std. Error    t value
           (Intercept)  -9.789761   81.03497 -0.1208091
   symmBaltimore,Boston  10.961595   81.03583  0.1352685
symmBaltimore,Cleveland  11.262285   81.03574  0.1389792
  symmBaltimore,Detroit  10.704983   81.03590  0.1321017
symmBaltimore,Milwaukee  10.586298   81.03594  0.1306371
       symmBaltimore,NY  10.854633   81.03586  0.1339485
  symmBaltimore,Toronto  10.818034   81.03587  0.1334969
etc…… snip.

                     Value Std. Error  t value
winningCleveland 0.6838528  0.3318767 2.060563
   winningBoston 1.1076977  0.3338782 3.317670
       winningNY 1.2476178  0.3358609 3.714686
  winningToronto 1.2944851  0.3366694 3.844974
  winningDetroit 1.4364084  0.3395685 4.230099
winningMilwaukee 1.5813559  0.3432560 4.606929

(Dispersion Parameter for Poisson family taken to be 1 )

    Null Deviance: 133.8648 on 48 degrees of freedom

Residual Deviance: 15.73729 on 15 degrees of freedom

Number of Fisher Scoring Iterations: 8
```

I added `winning` to the model in order to differentiate main effects for rows and columns. However, I reversed the levels of `winning` so that the last category coefficient (for Baltimore) is zeroed out. The resulting coefficient estimates are the differences $\left\{\lambda_j^Y - \lambda_j^X\right\}$ for j = 1, …, 6, which we see match those in Table 10.11 in Agresti.

Fitted counts are obtained in the same way

```
matrix(round(fitted(fit.BTQS),1),nr=7,nc=7,byrow=T,dimnames=list(win.team,
    losing.team))
```

J. Bradley-Terry Model with Order Effect

We would like to see if there is a significant home team advantage. That is, is a team more likely to win if they play in their home city? We can incorporate a home team advantage for the team called *a*, via a logit model for the probability that team *a* beats team *b* when *a* is the home team:

$$\log \frac{\Pi^*_{ab}}{1-\Pi^*_{ab}} = \alpha + (\beta_a - \beta_b)$$

When $\alpha > 0$, there is a home team advantage. If the teams were equally matched $(\beta_a = \beta_b)$, the home team has probability $\exp(\alpha)/[1+\exp(\alpha)]$ of winning.

As mentioned in Agresti, the 42 pair sets (where order matters) can be viewed as 42 independent binomial samples, where response is the number of successes (wins) for the home team out of the total played. So, for example,

```
response<-cbind(c(4,4,4,6,4,6,4,4,6,6,4,2,4,4,6,4,4,6,5,6,2,
                  3,2,3,5,2,2,4,5,2,3,1,2,3,3,1,4,4,2,4,1,3),
                 c(3,2,3,1,2,0,2,3,0,1,3,4,3,2,0,3,2,1,2,0,4,
                  3,5,3,1,5,5,3,1,5,3,5,5,3,4,6,2,3,4,2,6,4))       # 42 pair sets
```

Now, we use seven dummy variables that indicate by a 1, who is the home team for that pair set, and by a –1 who is the away team for the pair set.

```
Milwaukee<-c( 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1,-1,-1,-1,-1,-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
Detroit<-  c(-1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,-1,-1,-1,-1,-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
Toronto<-  c( 0,-1, 0, 0, 0, 0,-1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,-1,-1,-1,-1,-1, 0, 0, 0, 0, 0, 0)
NY<-       c( 0, 0,-1, 0, 0, 0, 0,-1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,-1,-1,-1,-1, 0, 0, 0, 0)
Boston<-   c( 0, 0, 0,-1, 0, 0, 0, 0,-1, 0, 0, 0,-1, 0, 0,-1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,-1,-1, 0)
Cleveland<-c( 0, 0, 0, 0,-1, 0, 0, 0, 0,-1, 0, 0, 0,-1, 0, 0,-1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0,-1)
Baltimore<-c( 0, 0, 0, 0, 0,-1, 0, 0, 0, 0,-1, 0, 0, 0,-1, 0, 0,-1, 0,-1,-1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1)
```

Then, we put all but the last category (Baltimore) in the `glm` call. The intercept is the estimate of α.

```
options(contrasts=c("contr.treatment", "contr.poly"))
fit.BTO<-glm(response~Milwaukee+Detroit+Toronto+NY+Boston+Cleveland,family=binomial)
summary(fit.BTO, cor=F)

Coefficients:
             Value Std. Error  t value
(Intercept) 0.2963483  0.1307495 2.266536
  Milwaukee 1.6436308  0.3473118 4.732435
    Detroit 1.4994142  0.3444591 4.352953
    Toronto 1.3512237  0.3402589 3.971164
         NY 1.3054814  0.3403238 3.835998
     Boston 1.1677444  0.3377718 3.457199
  Cleveland 0.7477615  0.3318849 2.253075

(Dispersion Parameter for Binomial family taken to be 1 )

    Null Deviance: 74.14759 on 41 degrees of freedom

Residual Deviance: 38.62303 on 35 degrees of freedom

Number of Fisher Scoring Iterations: 3
```

Thus, the estimate of the home team advantage parameter is positive, not surprisingly, implying a home team advantage, and its ASE is not too large.

The fitted probabilities are obtained using

```
fitted.probs<-matrix(0,nc=7,nr=7,dimnames=list(win.team,win.team))
fitted.probs[lower.tri(fitted.probs,diag=F) | !lower.tri(fitted.probs,diag=T)]<-
   round(fitted(fit.BTO),2)
```

```
          Milwaukee Detroit Toronto   NY Boston Cleveland Baltimore
Milwaukee      0.00    0.61    0.62 0.67   0.46      0.39      0.54
  Detroit      0.61    0.00    0.71 0.81   0.35      0.23      0.44
  Toronto      0.64    0.62    0.00 0.74   0.21      0.56      0.27
       NY      0.65    0.65    0.84 0.00   0.54      0.53      0.47
   Boston      0.68    0.74    0.61 0.54   0.00      0.42      0.29
Cleveland      0.77    0.86    0.70 0.50   0.53      0.00      0.39
Baltimore      0.87    0.58    0.83 0.49   0.49      0.26      0.00
```

These are not much different than those from the Bradley-Terry model without order effect.

Bradley-Terry models both with and without an order effect are easily fit using a new R package called `BradleyTerry` by D. Firth. In fact, the above example is illustrated in the help page for the function `BTm` in the package. Thus, I refer you to that page to see an easier way to fit these models in R.

K. Marginal and Quasi-symmetry Models for Matched Sets

A matched set generalizes a matched pair to three or more members or responses. Suppose there are three responses in a set, with each response being binary. Then, the three-way table has eight cells. The joint probabilities in the cells are $P(Y_1 = i_1, Y_2 = i_2, Y_3 = i_3) = \pi_{(i_1,i_2,i_3)}$ for $i_t = 0$ or 1. The marginal probabilities are $P(Y_1 = j) = \pi_{j++}$, $P(Y_2 = j) = \pi_{+j+}$, and $P(Y_3 = j) = \pi_{++j}$ for j = 0,1. Marginal homogeneity is satisfied when the marginal probabilities are equal for each j = 0,1. Complete symmetry is satisfied when $\pi_{(i_1,i_2,i_3)} = \pi_{(j_1,j_2,j_3)}$ for (j_1, j_2, j_3) a permutation of (i_1, i_2, i_3). For example, under complete symmetry, $P(Y_1 = 1, Y_2 = 1, Y_3 = 0) = P(Y_1 = 0, Y_2 = 1, Y_3 = 1)$.

As a loglinear model, complete symmetry is

$$\log \mu_{(i_1,i_2,i_3)} = \lambda_{abc} \tag{10.2}$$

where a is the minimum of (i_1, i_2, i_3), and c is the maximum. This model has $\binom{2+3-1}{3} = 4$ parameters when there are three binary responses. Quasi-symmetry adds "main effect" parameters

$$\log \mu_{(i_1,i_2,i_3)} = \lambda_{1i_1} + \lambda_{2i_2} + \lambda_{3i_3} + \lambda_{abc}$$

The identifiability constraints for a binary response are, for example, $\lambda_{10} = \lambda_{20} = \lambda_{30} = 0$.

When quasi-symmetry holds, in order for marginal homogeneity to hold, complete symmetry must be true. This can be seen by equating the marginal probabilities under the model (10.2), and realizing that they can't all be equal unless complete symmetry holds.

Agresti uses the Attitudes Toward Abortion example to illustrate fitting complete symmetry and quasi-symmetry models. Subjects indicated whether they supported legalized abortion in three situations (1) if

the family has a low income, (2) when the woman is not married and doesn't want to marry the man, and (3) when the woman wants it for any reason. Gender was also recorded. So, there are four binary responses.

Agresti first fits a complete symmetry model, with the same complete symmetry pattern within each gender. So, there is a different intercept for each gender. The data are read below. Notice the order of the levels of the question variables.

```
table.10.13<-data.frame(expand.grid(gender=factor(c("M","F"), levels=c("M","F")),
question1=factor(c("Y","N"), levels=c("N","Y")),
question2=factor(c("Y","N"), levels=c("N","Y")),
question3=factor(c("Y","N"), levels=c("N","Y"))),
count=c(342,440,6,14,11,14,19,22,26,25,21,18,32,47,356,457))
```

The symmetry factor is created a little differently when there are more than two responses, but the idea is the same. All permutations of the same set of categories are considered one level. Instead of using `pmin` and `pmax`, I use `sort` to get the permutations all the same. Actually, `sort` can be used even with only two responses. So, the following method is the most general method.

```
(temp<-apply(table.10.13[,c("question1","question2","question3")],1,function(x)
paste(sort(x),collapse=",")))
```

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

Then, I make the vector a factor, using levels "N,N,N" "N,N,Y" "N,Y,Y" and "Y,Y,Y", which causes `glm` to effectively set to zero any parameter with sequence that has N in the last slot.

```
table.10.13$symm<-factor(temp, levels=rev(unique(temp)))
```

Now, I fit the complete symmetry model, excluding an intercept so that each gender has its own intercept.

```
options(contrasts=c("contr.treatment", "contr.poly"))
glm(count~-1+gender+symm, family=poisson, data=table.10.13)

Coefficients:
  genderM    genderF  symmN,N,Y  symmN,Y,Y  symmY,Y,Y
  5.87852    6.12188   -2.73044   -3.23500   -0.03888

Degrees of Freedom: 16 Total (i.e. Null);  11 Residual
Null Deviance:      17080
Residual Deviance: 39.18        AIC: 137.9
```

The likelihood ratio statistic is 39.18, on 11 df. A quasi-symmetry model, with the same quasi-symmetry pattern per gender, fits better. Below is the fit using `glm`. The nonzero main effects are the Yes response on question 1 and the Yes response on question 2. The remaining are zero for identifiability purposes.

```
(fit.qs<-glm(count~-1+gender+question1+question2+symm, family=poisson,
data=table.10.13))

Coefficients:
   genderM     genderF  question1Y  question2Y   symmN,N,Y   symmN,Y,Y
    5.8785      6.1219      0.8284      0.3074     -3.1686     -4.0479

 symmY,Y,Y
   -1.1747

Degrees of Freedom: 16 Total (i.e. Null);  9 Residual
Null Deviance:      17080
```

```
Residual Deviance: 10.16        AIC: 112.8
```

To see what the model predicts, we can look at the fitted probabilities.

```
fitted.qs<-predict(fit.qs,type="response")/sum(table.10.13$count)
(fitted.table.10.13<-cbind(table.10.13,fitted.qs))
```

```
   gender question1 question2 question3 count  symm   fitted.qs
1       M         Y         Y         Y   342 Y,Y,Y 0.185760701
2       F         Y         Y         Y   440 Y,Y,Y 0.236942001
3       M         N         Y         Y     6 N,Y,Y 0.004585075
4       F         N         Y         Y    14 N,Y,Y 0.005848367
5       M         Y         N         Y    11 N,Y,Y 0.007720601
6       F         Y         N         Y    14 N,Y,Y 0.009847803
7       M         N         N         Y    19 N,N,Y 0.008123250
8       F         N         N         Y    22 N,N,Y 0.010361390
9       M         Y         Y         N    26 N,Y,Y 0.010498707
10      F         Y         Y         N    25 N,Y,Y 0.013391339
11      M         N         Y         N    21 N,N,Y 0.011046241
12      F         N         Y         N    18 N,N,Y 0.014089731
13      M         Y         N         N    32 N,N,Y 0.018600268
14      F         Y         N         N    47 N,N,Y 0.023725065
15      M         N         N         N   356 N,N,N 0.193124617
16      F         N         N         N   457 N,N,N 0.246334843
```

The highest probabilities occur for the two extremes. We can look at some marginal probabilities using `aggregate`, for example. Below are the distributions for each question, by gender.

```
aggregate(fitted.table.10.13$fitted.qs,
list(gender=fitted.table.10.13$gender,question1=fitted.table.10.13$question1), sum)
```

```
  gender question1         x
1      M         N 0.2168792
2      F         N 0.2766343
3      M         Y 0.2225803
4      F         Y 0.2839062
```

```
aggregate(fitted.table.10.13$fitted.qs,
list(gender=fitted.table.10.13$gender,question2=fitted.table.10.13$question2), sum)
```

```
  gender question2         x
1      M         N 0.2275687
2      F         N 0.2902691
3      M         Y 0.2118907
4      F         Y 0.2702714
```

```
aggregate(fitted.table.10.13$fitted.qs,
list(gender=fitted.table.10.13$gender,question3=fitted.table.10.13$question3), sum)
```

```
  gender question3         x
1      M         N 0.2332698
2      F         N 0.2975410
3      M         Y 0.2061896
4      F         Y 0.2629996
```

For question 1, “Yes” responses are predicted slightly more often than “No” responses for both men and women. This is not the case for the other two questions.

Chapter 11 – Analyzing Repeated Categorical Response Data

A. Summary of Chapter 11, Agresti

Repeated categorical responses may come from repeated measurements over time on each individual or from a set of measurements that are related because they belong to the same group or cluster (e.g., measurements made on siblings from the same family, measurements made on a set of teeth from the same mouth). Observations within a cluster are not usually independent of each other, as the response from one child of a family, say, may influence the response from another child, because the two grew up together. Matched-pairs are the special case of each cluster having two members.

Using repeated measures within a cluster can be an efficient way to estimate the mean response at each measurement time without estimating between-cluster variability. Many times, one is interested in the marginal distribution of the response at each measurement time, and not substantially interested in the correlation between responses across times. Estimation methods for marginal modeling include maximum likelihood estimation and generalized estimating equations. Maximum likelihood estimation is difficult because the likelihood is written in terms of the I^T multinomial joint probabilities for T responses with I categories each, but the model applies to the marginal probabilities. Lang and Agresti give a method for maximum likelihood fitting of marginal models in Section 11.2.5. Modeling a repeated multinomial response or repeated ordinal response is handled in the same way.

Generalized estimating equations (GEE) are a multivariate generalization of quasi-likelihood estimation. If there are T measurements per subject, then the quasi-likelihood method specifies a model for the mean and variance (as a function of the mean) for each measurement, without requiring knowledge of the distribution of the measurements (as ML estimation does). A "working" covariance structure for the responses must be specified. The estimates of parameters in the mean response are solutions of quasi-likelihood equations called generalized estimating equations. The estimates are consistent even if the covariance matrix is incorrectly specified. However, this does not imply that one can use an identity matrix for the correlation structure. The standard errors would be incorrect (probably underestimated). They are adjusted using the empirical dependence.

However, as GEE does not have a likelihood function, then inference procedures based on a likelihood cannot be done. Instead, inference only uses Wald statistics and their asymptotic normality.

Transitional models are special cases of logit or loglinear models for repeated categorical observations. When the repeated observations can be assumed to follow a Markov chain, an appropriate Markov chain model can be fit. The first-order Markov property is satisfied when the prediction of a future response only depends on the response immediately prior to it, and no other responses occurring before that previous response. Estimation of Markov chain models is via IRLS. Estimation can also be done using IPF if the model is a loglinear model. When there are explanatory variables, a regressive logistic model can be used (for a binary response) where the regressors are explanatory variables and also previous responses.

B. Comparing Marginal Distributions: Multiple Responses

With *T* repeated binary measurements, one may be interested in comparing the probability of success at each of the *T* times. The marginal logit model for *T* responses is

$$\text{logit}[P(Y_t = 1)] = \alpha + \beta_t, \qquad t = 1,...,T \tag{11.1}$$

Because there are *T* probabilities with *T* + 1 parameters, one of the parameters must be constrained to be zero. In that case, the model is saturated, and MLEs of the marginal probabilities are the sample proportions with $y_t = 1$. The multinomial likelihood is written in terms of the joint probabilities of response. Each sample cell proportion is the MLE of the joint probability corresponding to that cell. Marginal

homogeneity corresponds to $P(Y_1 = 1) = ... = P(Y_T = 1)$. That is, there are *T* identical response distributions across the *T* time periods. A likelihood ratio test compares the likelihood maximized under marginal homogeneity to the ML fit under model (11.1). Maximum likelihood estimation under marginal homogeneity is discussed in Section 11.2.5 in Agresti and in Section D in Chapter 10 of this manual, for illustrating Lang's `mph.fit` function. The generalized loglinear form of the marginal logit model is given in (11.8) of Agresti, and repeated in (10.1) of this manual. For the case of a 2^T table, there are *T* marginal probabilities. The model of marginal homogeneity with no covariates has the form given at the bottom of p. 464 in Agresti when *T* = 2. The Lagrangian includes constraints that set the marginal probabilities equal to one another, as well as identifiability constraints. The fitting method of Lang gives consistent estimates of the regression coefficients, β, if the marginal model holds, regardless of the form of the joint distribution. This is because the marginal parameters β and the joint parameters π are orthogonal. Thus, for consistency of the marginal parameter estimates, the method is robust to the specification of the joint distribution.

Agresti tests marginal homogeneity for the Crossover Drug Comparison Example (Table 11.1) where each subject used each of three drugs at three different times for the treatment of a chronic condition. The binary response at each time was favorable/unfavorable response. We first use the function `mph.fit` for R (available from J. Lang. See Agresti p. 649) to test marginal homogeneity.

First, I source in the appropriate files.

```
source("C:/Program Files/R/rw1080/CDA/mph.r")
source("C:/Program Files/R/rw1080/CDA/mph.supp.fcts.r")
```

I happen to set up the data as a data frame, but we only use the count vector within `mph.fit` for the test.

```
table.11.1<-data.frame(expand.grid( C=factor(c("Y","N"), levels=c("N","Y")),
                                    B=factor(c("Y","N"), levels=c("N","Y")),
                                    A=factor(c("Y","N"), levels=c("N","Y")),
                                    count=c(6,16,2,4,2,4,6,6))

y <- table.11.1$count
```

As in Chapter 10, we do not have stratification. So, the Z matrix is a vector of ones. Also ZF is a vector of ones because the sample size is assumed fixed.

```
Z <- matrix(1,2*2*2,1)
ZF <- Z
```

M1 describes the marginal totals for drug C. M2 describes the marginal totals for drug B, and M3 describes the marginal totals for drug A. The first argument to `Marg.fct` is the index to NOT sum over, the next argument is the number of levels of the factors (here, 2, 2, and 2).

```
M1 <- Marg.fct(1,rep(2,3))  # used to get y1++, etc
M2 <- Marg.fct(2,rep(2,3))  # used to get y+1+, etc
M3 <- Marg.fct(3,rep(2,3))  # used to get y++1, etc
```

The constraint matrix (specified by `C.matrix` below) represents the constraints implied by marginal homogeneity. As indicated in the definition, some constraints are redundant.

```
C.matrix <- matrix(c(
               1, 0, -1, 0, 0, 0,       # y1++ = y+1+
               0, 0, 1, 0, -1, 0),      # y+1+ = y++1
        2,6,byrow=T)
```

The function `h.fct` is the gradient of the Lagrangian.

```
h.fct <- function(m) { # constraint function
   marg <- rbind(M1%*%m,  M2%*%m, M3%*%m)  # y1++, y2++, y+1+, y+2+, y++1, y++2
   C.matrix%*%marg                         # y1++ = y+1+, y+1+ = y++1, etc
}
```

Now, we fit the model

```
a <- mph.fit(y=y,Z=Z,ZF=ZF,h.fct=h.fct)

mph.summary(a)
```

```
OVERALL GOODNESS OF FIT: TEST of   Ho: h(m)=0 vs. Ha: not Ho...
    Likelihood Ratio Stat (df= 2 ):  Gsq =  5.94507 (p =  0.051174 )
    Pearson's Score Stat  (df= 2 ):  Xsq =  5.71054 (p =  0.05754 )
    Generalized Wald Stat (df= 2 ):  Wsq =  5.76 (p =  0.056135 )


CONVERGENCE STATISTICS...
    iterations = 10
    norm.diff  = 3.46689e-06
    norm.score = 6.60356e-06
    Original counts used.

FITTING PROGRAM USED:  mph.fit, version 1.0, 6/5/02
```

which gives a LRT statistic of about 5.95 (p = 0.05, df = 2). The Generalized Wald statistic is the score statistic discussed in Section 11.1.4 of Agresti.

Bhapkar's statistic was computed in Chapter 10 Section D.2. The only change is for the matrix A, which is now

```
A<-matrix(c(0,0,1,1,-1,-1,0,0,
            0,1,-1,0,0,1,-1,0), nc=8,nr=2,byrow=T)
```

This matrix corresponds to equal marginal probabilities if the probability vector is $\boldsymbol{\pi} = (\pi_{111},\pi_{112},\pi_{121},\pi_{122},$ $\pi_{211},\pi_{212},\pi_{221},\pi_{222})$.

Multiple instead of binary responses are easily handled by making appropriate changes to the M matrices and `C.matrix`, as in Section D of Chapter 10.

The function `marg.hom` in the `repeated` library fits marginal homogeneity models to tables with two dimensions using maximum likelihood estimation.

C. Marginal Modeling: Maximum Likelihood Approach

The last section dealt with comparing marginal distributions and testing for homogeneity of those distributions. In this section, we will estimate the parameters of the marginal logit model with explanatory variables, using maximum likelihood estimation. We'll use the longitudinal mental depression data available on Agresti's CDA web site. This data set refers to a longitudinal study comparing a new drug with a standard drug for depression. Subjects were classified into two initial diagnosis groups (Mild, Severe). In each group, subjects were randomly assigned to one of the two drugs. At 1 week, 2 weeks, and 4 weeks, each subject's suffering from depression was classified as normal or abnormal.

As there are T = 3 occasions and binary responses, this is a 2^3 table at each of the treatment x initial diagnosis groupings. There are 12 marginal distributions, one at each of the response times for each of the groupings.

I have copied the appropriate data lines and column headers from Agresti's CDA data set web site, and pasted them into a file, which I've renamed "depress.txt". The time variable is transformed using log base 2.

```
table.11.2<-read.table("c:/program files/r/rw1080/cda/depress.txt", header=T)
table.11.2[1:10,]   # first 10 values
```

```
   case diagnose treat time outcome
1     1        0     0    0       1
2     1        0     0    1       1
3     1        0     0    2       1
4     2        0     0    0       1
5     2        0     0    1       1
6     2        0     0    2       1
7     3        0     0    0       1
8     3        0     0    1       1
9     3        0     0    2       1
10    4        0     0    0       1
```

Here are the sample marginal proportions

```
tapply(table.11.2$outcome,
list(diagnose=table.11.2$diagnose,treat=table.11.2$treat,time=table.11.2$time),   mean)
      # output is R
```

```
, , time = 0

         treat
diagnose      0         1
       0 0.5125 0.5285714
       1 0.2100 0.1777778

, , time = 1

         treat
diagnose      0         1
       0 0.5875 0.7857143
       1 0.2800 0.5000000

, , time = 2

         treat
diagnose     0         1
       0 0.675 0.9714286
       1 0.460 0.8333333
```

In Agresti's notation, s indexes the severity of the diagnosis (diagnose; Severe = 1, Mild = 0), t indexes time (0, 1, 2), and d indexes the drug (treat; New = 1, Standard = 0). Agresti considers a group to be a diagnose x treat combination. Let the variable Y_t be the outcome (Normal = 1; Abnormal = 0) at time t. Then, the marginal logit model with main effects and a linear effect of time that is the same within each group is

$$\text{logit}\left[P(Y_t = 1)\right] = \alpha + \beta_1 s + \beta_2 d + \beta_3 t \qquad (11.2)$$

The sampling model is product multinomial, with four groups, and eight cells per group (2^3 cross-classifications of the three responses), leading to 32 cell probabilities. However, (11.2) refers to marginal probabilities, and the three marginal binomial variates per group are dependent. As explained in Section B of this chapter, to fit this model using maximum likelihood estimation, we use the methodology from Section 11.2.5 in Agresti.

To begin, we define the indices g to index the four groups (1=Severe/New; 2=Severe/Standard, 3=Mild/New, 4=Mild/Standard). We also define the indices t_1 to t_3 to index the *outcome values* at the three measurement times. Then, we set the vector $\boldsymbol{\pi}$ equal to the 32 cell probabilities $\{\pi_{gt_1t_2t_3}\}$. For example, the probability of a normal outcome at all three times from an individual on the new drug, with severe depression is π_{1111}. The marginal probability of normal response for this individual at time 0 is the anti-logit of $\alpha+\beta_1+\beta_2$.

We define the **A** and **C** matrices to get the expression on the left of (11.2). For example, to write the marginal probability of Normal response at time 0, for group 1, we compute the sum of cell probabilities, $\sum_{t_3=0}^{1}\sum_{t_2=0}^{1}\pi_{1,1,t_2,t_3}$. Thus, the row of **A** that corresponds to this sum has a 1 in each of the spots that corresponds to each of the terms of the sum. Each row of the matrix **C** has exactly one element with a –1 and exactly one element with a +1, the remaining being 0. This matrix forms the logits (a ratio of logs of sums of cell probabilities). Finally, the matrix **X** has one column of all ones (for the intercept), two columns for the group designation, and a final column for the value of time (0, 1, 2). Using these definitions, we can run `mph.fit` for R.

First, we must "unstack" the data frame `table.11.2` so that time is three variables (log week 0, 1, and 2) instead of one.

```
table.11.2a<-data.frame(unique(table.11.2[,c("case","treat","diagnose")]),
unstack(table.11.2,outcome~time))
names(table.11.2a)[4:6]<-c("t1","t2","t3")

table.11.2a[1:6,]
```

```
   case treat diagnose t1 t2 t3
1     1     0        0  1  1  1
4     2     0        0  1  1  1
7     3     0        0  1  1  1
10    4     0        0  1  1  1
13    5     0        0  1  1  1
16    6     0        0  1  1  1
```

Now, we make the variables factors to help control the ordering of the levels. Here, I have the orderings all 1, 0. Either order is acceptable. However, the order of the vector of counts depends on the order you choose here.

```
table.11.2a$diagnose<-factor(table.11.2a$diagnose, levels=c(1,0))
table.11.2a$treat<-factor(table.11.2a$treat, levels=c(1,0))
table.11.2a$t1<-factor(table.11.2a$t1, levels=c(1,0))
table.11.2a$t2<-factor(table.11.2a$t2, levels=c(1,0))
table.11.2a$t3<-factor(table.11.2a$t3, levels=c(1,0))
```

Now, we form the 32 x 1 vector of counts in each cell. The order of the counts follows that given for the factors. Thus, the first count is for group 1: diagnose = 1 and treat = 1, for normal evaluations at all three times (i.e., t1=1, t2=1, t3=1).

```
Y<-c(tapply(table.11.2a$case,table.11.2a[,2:6],length))
Y<-ifelse(is.na(Y),0,Y)
Y
```

```
[1]  7  2 31 16 31  9 22 14  5  8  6  9 32 27  9 15  2  2  0 13  5 15  2  4  2
[26]  9  0  3  6 28  0  6
```

The design matrix has "diagnose" changing the slowest and "time" changing the fastest.

```
(X<-cbind(1,expand.grid(time=0:2, treat=c(1,0), diagnose=c(1,0) )))
```

```
   1 time treat diagnose
1  1    0     1        1
2  1    1     1        1
3  1    2     1        1
4  1    0     0        1
5  1    1     0        1
6  1    2     0        1
7  1    0     1        0
8  1    1     1        0
9  1    2     1        0
10 1    0     0        0
11 1    1     0        0
12 1    2     0        0
```

The sampling matrix has a column for each group and a row for each of the 32 cells, where the rows are in the same order as the **Y** vector. In each column, there is a 1 for each cell that corresponds to a cell for the group for that column.

```
Z<-kronecker(matrix(rep(1,8),nc=1),diag(4))
(ZF<-Z)
```

```
      [,1] [,2] [,3] [,4]
 [1,]    1    0    0    0
 [2,]    0    1    0    0
 [3,]    0    0    1    0
 [4,]    0    0    0    1
 [5,]    1    0    0    0
 [6,]    0    1    0    0
 [7,]    0    0    1    0
 [8,]    0    0    0    1
 [9,]    1    0    0    0
[10,]    0    1    0    0
[11,]    0    0    1    0
[12,]    0    0    0    1
[13,]    1    0    0    0
[14,]    0    1    0    0
[15,]    0    0    1    0
[16,]    0    0    0    1
[17,]    1    0    0    0
[18,]    0    1    0    0
[19,]    0    0    1    0
[20,]    0    0    0    1
[21,]    1    0    0    0
[22,]    0    1    0    0
[23,]    0    0    1    0
[24,]    0    0    0    1
[25,]    1    0    0    0
[26,]    0    1    0    0
[27,]    0    0    1    0
[28,]    0    0    0    1
[29,]    1    0    0    0
[30,]    0    1    0    0
[31,]    0    0    1    0
[32,]    0    0    0    1
```

Each row of the **A** matrix, when multiplied by the π vector, results in one of the marginal probabilities or its complement. For example, the first row of `A.matrix` below, when multiplied by π gives the probability that a randomly chosen observation from group 1 is evaluated as Normal at week (time t0). The fourth row gives its complement.

```
A.matrix<-matrix(c(1,0,0,0, 0,0,0,0, 1,0,0,0, 0,0,0,0, 1,0,0,0, 0,0,0,0, 1,0,0,0, 0,0,0,0, # g1 t0=1
                   1,0,0,0, 1,0,0,0, 0,0,0,0, 0,0,0,0, 1,0,0,0, 1,0,0,0, 0,0,0,0, 0,0,0,0, # g1 t1=1
                   1,0,0,0, 1,0,0,0, 1,0,0,0, 1,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, # g1 t2=1

                 # complement
```

```
    0,0,0,0, 1,0,0,0, 0,0,0,0, 1,0,0,0, 0,0,0,0, 1,0,0,0, 0,0,0,0, 1,0,0,0, # g1 t0=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, 1,0,0,0, 1,0,0,0, # g1 t1=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, 1,0,0,0, 1,0,0,0, # g1 t2=0

    0,1,0,0, 0,0,0,0, 0,1,0,0, 0,0,0,0, 0,1,0,0, 0,0,0,0, 0,1,0,0, 0,0,0,0, # g2 t0=1
    0,1,0,0, 0,1,0,0, 0,0,0,0, 0,0,0,0, 0,1,0,0, 0,1,0,0, 0,0,0,0, 0,0,0,0, # g2 t1=1
    0,1,0,0, 0,1,0,0, 0,1,0,0, 0,1,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, # g2 t2=1

# complement
    0,0,0,0, 0,1,0,0, 0,0,0,0, 0,1,0,0, 0,0,0,0, 0,1,0,0, 0,0,0,0, 0,1,0,0, # g2 t0=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, 0,1,0,0, 0,1,0,0, # g2 t1=0
    0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,1,0,0, 0,1,0,0, 0,1,0,0, 0,1,0,0, # g2 t2=0

    0,0,1,0, 0,0,0,0, 0,0,1,0, 0,0,0,0, 0,0,1,0, 0,0,0,0, 0,0,1,0, 0,0,0,0, # g3 t0=1
    0,0,1,0, 0,0,1,0, 0,0,0,0, 0,0,0,0, 0,0,1,0, 0,0,1,0, 0,0,0,0, 0,0,0,0, # g3 t1=1
    0,0,1,0, 0,0,1,0, 0,0,1,0, 0,0,1,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, # g3 t2=1

# complement
    0,0,0,0, 0,0,1,0, 0,0,0,0, 0,0,1,0, 0,0,0,0, 0,0,1,0, 0,0,0,0, 0,0,1,0, # g3 t0=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, 0,0,1,0, 0,0,1,0, # g3 t1=0
    0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,1,0, 0,0,1,0, 0,0,1,0, 0,0,1,0, # g3 t2=0

    0,0,0,1, 0,0,0,0, 0,0,0,1, 0,0,0,0, 0,0,0,1, 0,0,0,0, 0,0,0,1, 0,0,0,0, # g4 t0=1
    0,0,0,1, 0,0,0,1, 0,0,0,0, 0,0,0,0, 0,0,0,1, 0,0,0,1, 0,0,0,0, 0,0,0,0, # g4 t1=1
    0,0,0,1, 0,0,0,1, 0,0,0,1, 0,0,0,1, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, # g4 t2=1

# complement
    0,0,0,0, 0,0,0,1, 0,0,0,0, 0,0,0,1, 0,0,0,0, 0,0,0,1, 0,0,0,0, 0,0,0,1, # g4 t0=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, 0,0,0,1, 0,0,0,1, # g4 t1=0
    0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,1, 0,0,0,1, 0,0,0,1, 0,0,0,1), # g4 t2=0

 nr=24,nc=32, byrow=TRUE)
```

When multiplied by the log of the entries in $\mathbf{A}\boldsymbol{\pi}$, `C.matrix` gives the logits of the 12 marginal probabilities.

```
C.matrix<-kronecker(diag(4),matrix(c(
              1, 0, 0, -1, 0, 0,
              0, 1, 0, 0, -1, 0,
              0, 0, 1, 0, 0, -1),nr=3,nc=6,byrow=T))
```

The function `L.fct` computes the right hand side of equation 11.8 in Agresti.

```
L.fct <- function(m) {
   C.matrix%*%log(A.matrix%*%m)
}
```

Now, we put all the above in the call to `mph.fit`.

```
fit<-mph.fit(y=Y,Z=Z,ZF=ZF,X=as.matrix(X),L.fct=L.fct)

mph.summary(fit)
```

```
OVERALL GOODNESS OF FIT: TEST of   Ho: h(m)=0 vs. Ha: not Ho...
    Likelihood Ratio Stat (df= 8 ):  Gsq =  34.56933 (p =  3.2025e-05 )
    Pearson's Score Stat  (df= 8 ):  Xsq =  31.39204 (p =  0.00011963 )
    Generalized Wald Stat (df= 8 ):  Wsq =  31.0137 (p =  0.00013972 )


LINEAR PREDICTOR MODEL RESULTS...
           BETA StdErr(BETA)   Z-ratio      p-value
beta1 -0.5264734   0.14818314 -3.552856 3.810725e-04
beta2  0.9069316   0.09188559  9.870226 0.000000e+00
beta3  0.8866717   0.14024137  6.322469 2.574170e-10
beta4 -1.2457301   0.14377651 -8.664351 0.000000e+00

         OBS LINK     ML LINK StdErr(L) LINK RESID
link1  -1.53147637 -0.88553191 0.1431524 -3.5424542
link2   0.00000000  0.02139975 0.1114149 -0.1195605
link3   1.60943791  0.92833142 0.1456704  3.7208456
```

```
link4  -1.32492541 -1.77220372 0.1580641  1.8976785
link5  -0.94446161 -0.86527205 0.1129916 -0.4220890
link6  -0.16034265  0.04165961 0.1320449 -1.3442489
link7   0.11441035  0.36019844 0.1542620 -1.3096890
link8   1.29928298  1.26713007 0.1434996  0.1283795
link9   3.52636052  2.17406086 0.1851307  3.8787830
link10  0.05001042 -0.52647349 0.1481831  3.2436321
link11  0.35364004  0.38045817 0.1208956 -0.1390165
link12  0.73088751  1.28738984 0.1554322 -2.4992189


CONVERGENCE STATISTICS...
    iterations = 100
    norm.diff  = 0.371203
    norm.score = 0.000702893
    Original counts used.

FITTING PROGRAM USED:  mph.fit, version 1.0, 6/5/02

Warning message:
the condition has length > 1 and only the first element will be used in: if (a$L !=
"NA")
```

The warning message given is apparently a bug in the `mph.summary` function, but would not affect results. It can be eliminated by changing the line

```
if (a$L != "NA") {
```

to

```
if (!all(is.na(a$L))) {
```

in the `mph.summary` function.

Here we see that this model gives a LR statistics of around 34.6. A much smaller LR statistic comes from adding an interaction term in treatment and time, implying that the effect of time varies over time.

```
X$trtxtime<-X$treat*X$time

fit.ia<-mph.fit(y=Y,Z=Z,ZF=ZF,X=as.matrix(X),L.fct=L.fct)
mph.summary(fit.ia)

OVERALL GOODNESS OF FIT: TEST of   Ho: h(m)=0 vs. Ha: not Ho...
    Likelihood Ratio Stat (df= 7 ):  Gsq =  4.23174 (p =  0.75273 )
    Pearson's Score Stat  (df= 7 ):  Xsq =  4.09301 (p =  0.769 )
    Generalized Wald Stat (df= 7 ):  Wsq =  4.13354 (p =  0.76427 )


LINEAR PREDICTOR MODEL RESULTS...
            BETA StdErr(BETA)    Z-ratio      p-value
beta1 -0.04385680    0.1698035 -0.2582797 7.961910e-01
beta2  0.48265283    0.1163793  4.1472408 3.365061e-05
beta3 -0.05836195    0.2245202 -0.2599408 7.949095e-01
beta4 -1.29282486    0.1448600 -8.9246498 0.000000e+00
beta5  1.00700523    0.1848744  5.4469688 5.123545e-08

          OBS LINK     ML LINK StdErr(L) LINK RESID
link1  -1.53147637 -1.39504362 0.1788551 -0.7015335
link2   0.00000000  0.09461445 0.1261114 -0.5590772
link3   1.60943791  1.58427252 0.2039109  0.1306587
link4  -1.32492541 -1.33668166 0.1680944  0.0652769
link5  -0.94446161 -0.85402883 0.1103859 -0.4795432
link6  -0.16034265 -0.37137600 0.1523241  1.5646063
link7   0.11441035 -0.10221875 0.1809579  1.3826817
```

```
link8   1.29928298  1.38743932 0.1510355 -0.3417647
link9   3.52636052  2.87709738 0.2337484  1.3582751
link10  0.05001042 -0.04385680 0.1698035  0.6448222
link11  0.35364004  0.43879603 0.1201678 -0.4368103
link12  0.73088751  0.92144887 0.1647290 -1.0296818


CONVERGENCE STATISTICS...
    iterations = 100
    norm.diff  = 1.25822
    norm.score = 9.27206e-10
    Original counts used.

FITTING PROGRAM USED:  mph.fit, version 1.0, 6/5/02
```

The beta coefficients correspond to the columns of the **X** matrix. Thus, at time *t*, the estimated odds of Normal response for the new drug are exp(–0.06 + 1.01*t*) times the odds for the standard drug, for each diagnosis level.

D. Marginal Modeling: Maximum Likelihood Approach. Modeling a Repeated Multinomial Response

With a repeated multinomial response with I categories, there are I – 1 logits for each observation or measurement. For observation *t*, the logit takes the form

$$\text{logit}_j(t) = \alpha_j + \boldsymbol{\beta}_j^T \mathbf{x}_t, \quad j = 1,...,I-1 \tag{11.3}$$

For example, for an ordinal response, a proportional odds model uses $\text{logit}_j(t) = \text{logit}\left[P(Y_t \le j)\right]$, a cumulative link, and $\boldsymbol{\beta}_j = \boldsymbol{\beta}$.

The data in Table 11.4 in Agresti give the results of a randomized clinical trial comparing an active hypnotic drug with a placebo in patients who have insomnia. The response is the reported time to fall asleep (with categories < 20 minutes, 20 – 30 minutes, 30 – 60 minutes, or > 60 minutes). For this data set, I copied the data from Agresti's website, pasted it into a text file called "insomnia.txt", and added a first line of variable headings. We will not be using the final variable called "count" in this data set.

We read in the data set using **read.table**.

```
table.11.4<-read.table("c:/program files/r/rw1080/cda/insomnia.txt", header=T)
```

To get the sample marginal distributions of the responses within each treatment/occasion group, we can use `tapply` and `xtabs`. First, we get the appropriate denominators for the proportions. The denominators will be the number of subjects per group. Here, we calculate the denominators, then expand them into an array the same size as the marginal table.

```
n<-array(tapply(table.11.4$case,list(treat=table.11.4$treat, time=table.11.4$time),
length), dim=c(2,4,2))
```

Next, we want the array to have each face with the same sample size. So, we permute the array using `aperm`.

```
n<-aperm(n, c(3,2,1))
n

, , 1
```

```
     [,1] [,2] [,3] [,4]
[1,]  120  120  120  120
[2,]  120  120  120  120

, , 2

     [,1] [,2] [,3] [,4]
[1,]  119  119  119  119
[2,]  119  119  119  119
```

Now, we get the marginal counts using `xtabs`, and divide these counts by `n`, which is an array of the same size (thus the division can be done).

```
xtabs(~time+outcome+treat, data=table.11.4)/n        # note that we divide by n

, , treat = 0

    outcome
time 1          2          3          4
   0 0.11666667 0.16666667 0.29166667 0.42500000
   1 0.25833333 0.24166667 0.29166667 0.20833333

, , treat = 1

    outcome
time 1          2          3          4
   0 0.10084034 0.16806723 0.33613445 0.39495798
   1 0.33613445 0.41176471 0.15966387 0.09243697
```

We next fit the proportional odds model in equation (11.6) in Agresti, using `mph.fit`. This model permits interaction between occasion and treatment.

The data set has a four-category (I = 4) ordinal response at T = 2 occasions. We follow the same steps as for section C. First, we "unstack" the data frame `table.11.4` so that time is two variables (initial, follow-up) instead of one.

```
table.11.4a<-data.frame(unique(table.11.4[,c("case","treat")]),
unstack(table.11.4,outcome~time))
names(table.11.4a)[3:4]<-c("initial","followup")

> table.11.4a[1:10,]

   case treat initial followup
1     1     1       1        1
3     2     1       1        1
5     3     1       1        1
7     4     1       1        1
9     5     1       1        1
11    6     1       1        1
13    7     1       1        1
15    8     1       1        2
17    9     1       1        2
19   10     1       1        2
```

Now, we make the variables factors to help control the ordering of the levels.

```
table.11.4a$treat<-factor(table.11.4a$treat, levels=c(1,0))
table.11.4a$initial<-factor(table.11.4a$initial, levels=1:4)
table.11.4a$followup<-factor(table.11.4a$followup, levels=1:4)
```

Now, we form the 32 x 1 vector of counts in each cell. The order of the counts follows that given for the levels of the factors, with treat levels changing fastest, and followup changing slowest. Thus, the first

count is for treat=1: active treatment, for initial and follow-up time < 20 minutes (coded 1). The second count is for treat=0, with the same initial and follow-up time.

```
Y<-c(tapply(table.11.4a$case,table.11.4a[,2:4],length))
Y<-ifelse(is.na(Y),0,Y)
Y
[1]  7  7 11 14 13  6  9  4  4  4  5  5 23  9 17 11  1  2  2  1  3 18 13 14  0
[26]  1  2  0  1  2  8 22
```

The design matrix has “time” changing the slowest and “treat” changing the fastest. The first three columns correspond to the category cut points (<20, 20-30, 30-60 minutes)

```
X<-data.frame(kronecker(diag(3),rep(1,4)),treat=1:0,time=c(0,0,1,1))
X$trtxtime<-X$time*X$treat
```

```
   X1 X2 X3 treat time trtxtime
1   1  0  0     1    0        0
2   1  0  0     0    0        0
3   1  0  0     1    1        1
4   1  0  0     0    1        0
5   0  1  0     1    0        0
6   0  1  0     0    0        0
7   0  1  0     1    1        1
8   0  1  0     0    1        0
9   0  0  1     1    0        0
10  0  0  1     0    0        0
11  0  0  1     1    1        1
12  0  0  1     0    1        0
```

The sampling matrix has a column for each group and a row for each of the 32 cells, where the rows are in the same order as the **Y** vector. In each column, there is a 1 for each cell that corresponds to a cell for the group for that column.

```
Z<-kronecker(matrix(rep(1,16),nc=1),diag(2))
(ZF<-Z)
```

```
      [,1] [,2]
 [1,]    1    0
 [2,]    0    1
 [3,]    1    0
 [4,]    0    1
 [5,]    1    0
 [6,]    0    1
 [7,]    1    0
 [8,]    0    1
 [9,]    1    0
[10,]    0    1
[11,]    1    0
[12,]    0    1
[13,]    1    0
[14,]    0    1
[15,]    1    0
[16,]    0    1
[17,]    1    0
[18,]    0    1
[19,]    1    0
[20,]    0    1
[21,]    1    0
[22,]    0    1
[23,]    1    0
[24,]    0    1
[25,]    1    0
[26,]    0    1
[27,]    1    0
```

```
[28,]    0    1
[29,]    1    0
[30,]    0    1
[31,]    1    0
[32,]    0    1
```

Each row of the **A** matrix, when multiplied by the π vector, results in one of the cumulative marginal probabilities or its complement. For example, the first row of `A.matrix` below, when multiplied by π gives the probability that a randomly chosen observation from group 1 (active) takes < 20 minutes to fall asleep at initial time. The third row gives its complement.

```
A.matrix<-matrix(c(1,0,0,0, 0,0,0,0, 1,0,0,0, 0,0,0,0, 1,0,0,0, 0,0,0,0, 1,0,0,0, 0,0,0,0, # g1 t1=<1
                   1,0,1,0, 1,0,1,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, # g1 t2=<1
# complement
                   0,0,1,0, 1,0,1,0, 0,0,1,0, 1,0,1,0, 0,0,1,0, 1,0,1,0, 0,0,1,0, 1,0,1,0, # g1 t1>1
                   0,0,0,0, 0,0,0,0, 1,0,1,0, 1,0,1,0, 1,0,1,0, 1,0,1,0, 1,0,1,0, 1,0,1,0, # g1 t2>1

                   1,0,1,0, 0,0,0,0, 1,0,1,0, 0,0,0,0, 1,0,1,0, 0,0,0,0, 1,0,1,0, 0,0,0,0, # g1 t1=<2
                   1,0,1,0, 1,0,1,0, 1,0,1,0, 1,0,1,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, # g1 t2=<2
# complement
                   0,0,0,0, 1,0,1,0, 0,0,0,0, 1,0,1,0, 0,0,0,0, 1,0,1,0, 0,0,0,0, 1,0,1,0, # g1 t1>2
                   0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 1,0,1,0, 1,0,1,0, 1,0,1,0, 1,0,1,0, # g1 t2>2

                   1,0,1,0, 1,0,0,0, 1,0,1,0, 1,0,0,0, 1,0,1,0, 1,0,0,0, 1,0,1,0, 1,0,0,0, # g1 t1=<3
                   1,0,1,0, 1,0,1,0, 1,0,1,0, 1,0,1,0, 1,0,1,0, 1,0,1,0, 0,0,0,0, 0,0,0,0, # g1 t2=<3
# complement
                   0,0,0,0, 0,0,1,0, 0,0,0,0, 0,0,1,0, 0,0,0,0, 0,0,1,0, 0,0,0,0, 0,0,1,0, # g1 t1>3
                   0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 1,0,1,0, 1,0,1,0, # g1 t2>3


                   0,1,0,0, 0,0,0,0, 0,1,0,0, 0,0,0,0, 0,1,0,0, 0,0,0,0, 0,1,0,0, 0,0,0,0, # g2 t1=<1
                   0,1,0,1, 0,1,0,1, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, # g2 t2=<1
# complement
                   0,0,0,1, 0,1,0,1, 0,0,0,1, 0,1,0,1, 0,0,0,1, 0,1,0,1, 0,0,0,1, 0,1,0,1, # g2 t1>1
                   0,0,0,0, 0,0,0,0, 0,1,0,1, 0,1,0,1, 0,1,0,1, 0,1,0,1, 0,1,0,1, 0,1,0,1, # g2 t2>1

                   0,1,0,1, 0,0,0,0, 0,1,0,1, 0,0,0,0, 0,1,0,1, 0,0,0,0, 0,1,0,1, 0,0,0,0, # g2 t1=<2
                   0,1,0,1, 0,1,0,1, 0,1,0,1, 0,1,0,1, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, # g2 t2=<2
# complement
                   0,0,0,0, 0,1,0,1, 0,0,0,0, 0,1,0,1, 0,0,0,0, 0,1,0,1, 0,0,0,0, 0,1,0,1, # g2 t1>2
                   0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,1,0,1, 0,1,0,1, 0,1,0,1, 0,1,0,1, # g2 t2>2

                   0,1,0,1, 0,1,0,0, 0,1,0,1, 0,1,0,0, 0,1,0,1, 0,1,0,0, 0,1,0,1, 0,1,0,0, # g2 t1=<3
                   0,1,0,1, 0,1,0,1, 0,1,0,1, 0,1,0,1, 0,1,0,1, 0,1,0,1, 0,0,0,0, 0,0,0,0, # g2 t2=<3
# complement
                   0,0,0,0, 0,0,0,1, 0,0,0,0, 0,0,0,1, 0,0,0,0, 0,0,0,1, 0,0,0,0, 0,0,0,1, # g2 t1>3
                   0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,1,0,1, 0,1,0,1),# g2 t2>3

 nr=24,nc=32, byrow=TRUE)
```

When multiplied by the log of the entries in $\mathbf{A}\pi$, `C.matrix` gives the cumulative logits of the 12 marginal probabilities.

```
C.matrix<-matrix(
c(1,0, -1,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0,  # P(time1 =< 1 for group1)
  0,0,  0,0, 0,0, 0,0, 0,0, 0,0, 1,0, -1,0, 0,0, 0,0, 0,0, 0,0, # P(time1 =< 1 for group2)

  0,1,  0,-1, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, # P(time2 =< 1 for group1)
  0,0,  0,0, 0,0, 0,0, 0,0, 0,0, 0,1, 0,-1, 0,0, 0,0, 0,0, 0,0, # P(time2 =< 1 for group2)

  0,0,  0,0, 1,0, -1,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, # P(time1 =< 2 for group1)
  0,0,  0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 1,0, -1,0, 0,0, 0,0, # P(time1 =< 2 for group2)

  0,0,  0,0, 0,1,  0,-1, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, # P(time2 =< 2 for group1)
  0,0,  0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,1, 0,-1, 0,0, 0,0,  # P(time2 =< 2 for group2)

  0,0,  0,0, 0,0,  0,0, 1,0, -1,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, # P(time1 =< 3 for group1)
  0,0,  0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 1,0, -1,0,  # P(time1 =< 3 for group2)

  0,0,  0,0, 0,0,  0,0, 0,1,  0,-1, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, # P(time2 =< 3 for group1)
  0,0,  0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,1, 0,-1),  # P(time2 =< 3 for group2)
```

```
nr=12,nc=24,byrow=T)
```

The function `L.fct` computes the right hand side of equation 11.8 in Agresti.

```
L.fct <- function(m) {
   C.matrix%*%log(A.matrix%*%m)
}
```

Now, we put all the above in the call to `mph.fit`.

```
fit<-mph.fit(y=Y,Z=Z,ZF=ZF,X=as.matrix(X),L.fct=L.fct)

mph.summary(fit)
```

```
OVERALL GOODNESS OF FIT: TEST of   Ho: h(m)=0 vs. Ha: not Ho...
    Likelihood Ratio Stat (df= 6 ):  Gsq =  8.0134 (p =  0.23712 )
    Pearson's Score Stat  (df= 6 ):  Xsq =  8.18076 (p =  0.22516 )
    Generalized Wald Stat (df= 6 ):  Wsq =  7.41296 (p =  0.28434 )


LINEAR PREDICTOR MODEL RESULTS...
            BETA StdErr(BETA)     Z-ratio      p-value
beta1 -2.27063692    0.2051665 -11.0672891 0.000000e+00
beta2 -0.95753227    0.1763571  -5.4295094 5.650918e-08
beta3  0.31961137    0.1737922   1.8390437 6.590876e-02
beta4  0.04605101    0.2363701   0.1948258 8.455293e-01
beta5  1.07393199    0.1624436   6.6111065 3.814571e-11
beta6  0.66226765    0.2438234   2.7161780 6.604040e-03

         OBS LINK    ML LINK StdErr(L) LINK RESID
link1  -2.1879222 -2.2245859 0.2045677  0.1583699
link2  -2.0243818 -2.2706369 0.2051665  1.0392564
link3  -0.6805684 -0.4883863 0.1740899 -2.6272947
link4  -1.0546492 -1.1967049 0.1748117  1.1160129
link5  -1.0001722 -0.9114813 0.1806545 -0.9644973
link6  -0.9279868 -0.9575323 0.1763571  0.2886920
link7   1.0874390  0.8247184 0.1749472  2.7610532
link8   0.0000000  0.1163997 0.1696301 -1.7029460
link9   0.4265185  0.3656624 0.1731676  0.8818240
link10  0.3022809  0.3196114 0.1737922 -0.2744260
link11  2.2842360  2.1018620 0.2054380  0.8657007
link12  1.3350011  1.3935434 0.1831185 -0.4272090


CONVERGENCE STATISTICS...
    iterations = 100
    norm.diff  = 1.28461
    norm.score = 5.21507e-09
    Original counts used.

FITTING PROGRAM USED:  mph.fit, version 1.0, 6/5/02
```

The MLE's of the coefficients follow the order of columns in the design matrix, X. So, the first three betas are the cutoff points. Beta4 is the treatment coefficient; beta5 is the occasion coefficient; and beta6 is the occasion by treatment interaction coefficient. The interaction coefficient is more than twice its standard error. As Agresti points out, at the initial time, the estimated odds that time to falling asleep for the active treatment is below any fixed level is exp(0.046) = 1.04 times the estimated odds for the placebo group. However, at the follow-up time, the estimated odds for the active treatment are exp(0.046 + 0.662) = 2.03 times that for the placebo group. So, by the follow-up time, those who took the active treatment tend to fall asleep more quickly.

E. Marginal Modeling: GEE Approach. Modeling a Repeated Multinomial Response

GEE methodology is a generalization of quasi-likelihood for multivariate responses. As with quasi-likelihood, one specifies the mean for the set of responses and a link function relating the linear predictor to the mean, as well as the variance as a function of the mean and a guess at the correlation structure among the responses. Estimation is accomplished via a set of estimating equations, yielding consistent estimates of the coefficients in the linear predictor regardless of the correctness of the correlation structure. However, the standard errors of the estimates will be affected by the specification of the correlation structure. Thus, it is advantageous to try to specify an approximately accurate correlation structure.

Common correlation structures are independence (no correlation), exchangeable (all pairs of responses have the same correlation), and unstructured. The unstructured correlation matrix requires the most new parameters to be estimated, and will not be efficient if there are many responses.

GEE differs from ML estimation in that no multivariate distribution has to be specified for the multivariate response. Thus, there is no likelihood with GEE. However, this also means there is no likelihood-based inference. Instead, Wald statistics and their asymptotic distributions are used.

Agresti uses GEE to fit a model to the Longitudinal Mental Depression Example from Table 11.2 (Section C, this chapter). First, we read in the data, and change the binary variables to factors.

```
# S-PLUS or R
table.11.2<-read.table("c:/program files/r/rw1080/cda/depress.txt", header=T)
table.11.2b<-table.11.2
table.11.2b$diagnose<-ifelse(table.11.2b$diagnose==0,"mild","severe")
table.11.2b$diagnose<-factor(table.11.2b$diagnose, levels=c("mild","severe"))
table.11.2b$treat<-ifelse(table.11.2b$treat==0,"standard","new")
table.11.2b$treat<-factor(table.11.2b$treat,levels=c("standard","new"))
table.11.2b<-table.11.2b[order(table.11.2b$case),]
```

Here are the first eight individuals.

```
table.11.2b[1:24,]
```

```
     case diagnose    treat time outcome
1       1     mild standard    0       1
2       1     mild standard    1       1
3       1     mild standard    2       1
4       2     mild standard    0       1
5       2     mild standard    1       1
6       2     mild standard    2       1
7       3     mild standard    0       1
8       3     mild standard    1       1
9       3     mild standard    2       1
10      4     mild standard    0       1
11      4     mild standard    1       1
12      4     mild standard    2       1
13      5     mild standard    0       1
14      5     mild standard    1       1
15      5     mild standard    2       1
16      6     mild standard    0       1
17      6     mild standard    1       1
18      6     mild standard    2       1
19      7     mild standard    0       1
20      7     mild standard    1       1
21      7     mild standard    2       1
22      8     mild standard    0       1
23      8     mild standard    1       1
24      8     mild standard    2       1
```

S-PLUS and R have libraries `yags` and `gee`, which do GEE analyses. R also has package `geepack`. These are illustrated below.

The version of `yags` for S-PLUS 6.1 does not have a `data` argument. Thus, it is necessary to attach the data set.

```
library(yags)
attach(table.11.2b) # S-PLUS only
options(contrasts=c("contr.treatment", "contr.poly"))
fit.yags<-yags(outcome~diagnose+ treat*time, id=case, cor.met=I(2^time),
   corstr="exchangeable", family=binomial, alphainit=.05)
# R: fit.yags<-yags(outcome~diagnose + time*treat, id=case, cor.met=I(2^time),
   family=binomial, corstruct="exchangeable", data=table.11.2b, alphainit=.05)
```

The argument `id` gives the variable that holds the individual id's. The argument cor.met gives the time scale. The time scale is in weeks (i.e., 1, 2, 4 weeks), which is the antilog base 2 of the time variable in the model equation. The argument `corstr` (`corstruct` in R) has options "independence", "exchangeable", "ar1", and "unstructured". The argument `family` indicates the mean function and the variance function as a relation to the mean. For the function `yags`, it is also used to get starting values from `glm`.

Summary results are obtained using `summary` (S-PLUS) or implicit `print` (R). Output is from S-PLUS.

```
summary(fit.yags)   # S-PLUS
fit.yags      # R


Coefficients:
              Estimate Naive S.E.    Naive z Robust S.E.   Robust z
(Intercept) -0.02809866  0.1625499 -0.1728617   0.1741791 -0.1613205
   diagnose -1.31391033  0.1448627 -9.0700417   0.1459630 -9.0016667
      treat -0.05926689  0.2205340 -0.2687427   0.2285569 -0.2593091
       time  0.48246420  0.1141154  4.2278625   0.1199383  4.0226037
 treat:time  1.01719312  0.1877051  5.4191017   0.1877014  5.4192084

Estimated Scale Parameter:  0.985392
Number of Iterations:  2

Working Correlation Parameter(s)
             [,1]
[1,] -0.003432732
```

Here we fit an exchangeable correlation structure. The estimate of the common correlation is very close to zero, indicating almost no correlation among responses at the various observation times. The parameter estimates are very close to the MLE's from Section C of this chapter. The `Robust S.E.`'s (`sand s.e.` in R) correspond to the square roots of the "sandwich covariance estimator" on p. 471 of Agresti. The standard errors are "robust" to misspecification of the variance function, as a function of the mean. The `Robust z`'s use the `Robust S.E.`'s. The `Naïve S.E.`s come from the asymptotic covariance matrix in equation (11.10) in Agresti.

The library `gee` gives similar output. Here I show the output from S-PLUS.

```
library(gee)
attach(table.11.2b) # S-PLUS only
options(contrasts=c("contr.treatment", "contr.poly"))
fit.gee<-gee(outcome~diagnose+ treat*time, id=case, family=binomial,
   corstr="exchangeable")
# R: fit.gee<-gee(outcome~diagnose + time*treat, id=case, family=binomial,
   corstr="exchangeable", data=table.11.2b)
summary(fit.gee)


GEE:  GENERALIZED LINEAR MODELS FOR DEPENDENT DATA
```

```
 gee S-function, version 5.2 modified 99/06/03 (1998)

Model:
 Link:                      Logit
 Variance to Mean Relation: Binomial
 Correlation Structure:     Exchangeable


Coefficients:
              Estimate Naive S.E.    Naive z Robust S.E.   Robust z
(Intercept) -0.02809866  0.1625499 -0.1728617   0.1741791 -0.1613205
   diagnose -1.31391033  0.1448627 -9.0700418   0.1459630 -9.0016667
      treat -0.05926689  0.2205340 -0.2687427   0.2285569 -0.2593091
       time  0.48246420  0.1141154  4.2278625   0.1199383  4.0226037
 treat:time  1.01719312  0.1877051  5.4191018   0.1877014  5.4192084

Estimated Scale Parameter:  0.985392
Number of Iterations:  2

Working Correlation
             [,1]         [,2]         [,3]
[1,]  1.000000000 -0.003432732 -0.003432732
[2,] -0.003432732  1.000000000 -0.003432732
[3,] -0.003432732 -0.003432732  1.000000000
```

R also has a package called `geepack`, with largely identical arguments. However, it allows different link functions for the mean, scale, and correlation. That is, the scale and correlation can depend on covariates, being related to them via link functions.

```
library(geepack)
fit.geese<-geese(outcome~diagnose + time*treat, id=case, data=table.11.2b,
family=binomial, corstr="exch")
summary(fit.geese)


Mean Model:
 Mean Link:                 logit
 Variance to Mean Relation: binomial

 Coefficients:
                  estimate    san.se        wald            p
(Intercept)    -0.02809866 0.1741402  0.02603594 8.718126e-01
diagnosesevere -1.31391033 0.1450823 82.01676945 0.000000e+00
time            0.48246420 0.1205994 16.00442887 6.319448e-05
treatnew       -0.05926689 0.2288857  0.06704815 7.956842e-01
time:treatnew   1.01719312 0.1881735 29.22066021 6.458684e-08

Scale Model:
 Scale Link:                identity

 Estimated Scale Parameters:
             estimate     san.se     wald p
(Intercept) 0.9805616 0.06497889 227.7221 0

Correlation Model:
 Correlation Structure:     exch
 Correlation Link:          identity

 Estimated Correlation Parameters:
         estimate     san.se       wald         p
alpha -0.003432732 0.03007298 0.01302948 0.9091215

Returned Error Value:    0
Number of clusters:   340   Maximum cluster size: 3
```

The output only gives the robust s.e.s (called `san.se`, here).

F. Marginal Modeling: GEE Approach. Modeling a Repeated Multinomial Ordinal Response

For repeated ordinal responses, GEE can also be used to estimate parameters from the logit model. For example, for the the proportional odds cumulative logit model, we get parameter estimates for the cutpoints, as well as the coefficients for the covariates. We can also get parameter estimates for the association parameters in the covariance matrix, relating the responses to one another.

The estimating equations are similar to those for univariate responses. The general form is given in section 11.4.4 of Agresti for multinomial responses. For ordinal responses in particular, the response variable $\mathbf{y}_i$ changes. Suppose we have an ordinal response with I = 4 categories and two responses per individual. Then, each $\mathbf{y}_i$ is length 2 x (4 – 1) = 6, with elements $(y_{i1}(1), y_{i1}(2), ..., y_{i2}(3))$. The element $y_{it}(1)$ is equal to 1 if the *t*th response for individual *i* is *greater than* 1. The element $y_{it}(2)$ equals 1 if the the *t*th response for individual *i* is greater than 2, etc. Thus, an individual with responses 3 and 4 at times 1 and 2 has $\mathbf{y}_i = (1,1,0,1,1,1)$.

For the proportional odds model, the vector $\boldsymbol{\mu}_i$ gives the probabilities associated with $\mathbf{y}_i$. Thus, for the example, $\boldsymbol{\mu}_i = (P(y_{i1} > 1), ..., P(y_{i2} > 3))$, where $P(y_{it} > j) = \left(1 + \exp(\alpha_j + \mathbf{x}_i^T \boldsymbol{\beta})\right)^{-1}$. Also, the covariance structure of the elements of $\mathbf{y}_i$ is put into $\mathbf{V}_i$. Ordinary GEE (GEE1) uses the estimating equations in section 11.4.4 of Agresti to estimate the regression coefficients and cut-points, using something like Fisher scoring (with intial estimates for the covariance parameters). Then to estimate the covariance parameters, a method of moments update is done, using the Fisher scoring estimates for the regression coefficients. These two steps are repeated until "convergence". The "sandwich" estimator of the covariance matrix for the regression coefficients is given in section 11.4.4 of Agresti.

A different estimation procedure, called GEE2, builds a set of estimating equations for the covariance parameters too, updating both sets of parameters simultaneously. Differences between the two estimation methods are found in the Heagerty and Zeger (1996) reference in Agresti. Certain R and S-PLUS functions (namely, `ordgee` in package `geepack`, and the function `ord.EE` for S-PLUS Unix) claim to use "the" method of Heagerty and Zeger, which is apparently GEE1, although Heagerty and Zeger compare *both* GEE1 and GEE2. According to computations from these functions and from SAS's GENMOD, they give different estimates than SAS does. I illustrate this for the insomnia data from table 11.4 in Agresti, that was analyzed in Section D of this chapter.

The function `ordgee` requires the data to be sorted by individual and response within individual. So, after reading in the data, we modify `table.11.4` as follows.

```
table.11.4ord<-table.11.4[order(table.11.4$case, table.11.4$time),]
```

Then, we make the outcome an ordered factor.

```
table.11.4ord$outcome<-ordered(table.11.4ord$outcome, levels=1:4)
```

The arguments for `ordgee` are similar to those for `geese`, as they are from the same package. Differences include the argument `rev`, which stands for reverse. Setting it to TRUE means that we want the cutpoints to correspond to the cumulative probabilities, $P(y_{it} \le j)$, not 1 minus the cumulative probabilities. Here, we fit the independence covariance structure to compare with Agresti's SAS GENMOD results.

```
library(geepack)
```

```
fit.ordgee<-ordgee(outcome~treat*time, id=case , data = table.11.4ord, corstr =
       "independence", rev=TRUE, control = geese.control(maxit = 100))    # 100
       iterations are specified in order to ensure convergence
summary(fit.ordgee)

 Mean Model:
 Mean Link:                 logit
 Variance to Mean Relation: binomial

 Coefficients:
             estimate    san.se        wald            p
Inter:1    -1.8912888 0.3866288 23.92914818 9.994697e-07
Inter:2    -0.6951357 0.3885746  3.20029850 7.362483e-02
Inter:3     0.4838944 0.4026376  1.44435030 2.294367e-01
treat       0.1810109 0.6292888  0.08273884 7.736196e-01
time        1.2445189 0.6099388  4.16323099 4.131053e-02
treat:time -0.9370106 0.9445051  0.98419339 3.211657e-01

Scale is fixed.

Correlation Model:
 Correlation Structure:     independence

Returned Error Value:    0
Number of clusters:   239   Maximum cluster size: 2
```

Although the cutpoint estimates are roughly similar to those from SAS (–2.26, –0.95, and 0.352, respectively), and the time coefficient is somewhat similar across the two programs, the interaction parameter estimate is negative in `ordgee`, but positive in SAS, which may indicate a coding difference (however, I have not been able to find it). Also, the standard errors are much larger in `ordgee`.

For S-PLUS version 6.2 and above, the `correlatedData` library (available from the Insightful website) can be used to fit marginal models to repeated ordinal data. The function `geeDesign` sets up the design of the model, and `gee.fit` is used to do the actual fit.

In the call to `geeDesign`, we give the formula for the fixed effects structure, and specify the cluster variable and the family to be multinomial. Then, to get an ordinal model, we give the link argument as "`clogit`" for cumulative logit. The `variance` argument can be used to specify a design for the variance of the ordinal response. Here, we let it follow the structure of a generalized linear model with a multiplicative of a scale parameter. Thus, the variance of the response is proportional to a function of the conditional mean, where the proportionality constant is the dispersion parameter.

```
library(correlatedData)
geedes<-geeDesign(outcome~treat*time, cluster=case, family="multinomial",
   link="clogit", data=table.11.4, variance="glm.scale")
gee.fit(geedes)

GEE: Generalized Estimating Equations
Model:
 Family:   multinomial
 Link  :   clogit

Estimated Parameters:
  Regression Coefficients:
             Estimate  Std.Err.      Z      Prob
         1 -2.2670890 0.2187605 -10.36 0.0000000
         2 -0.9514617 0.1809171  -5.26 0.0000001
         3  0.3517398 0.1784232   1.97 0.0486805
     treat  0.0336100 0.2384374   0.14 0.8879020
      time  1.0380764 0.1675855   6.19 0.0000000
treat:time -0.7077588 0.2435197  -2.91 0.0036565
```

```
  Scale Parameter:
1.013369

Number of iterations :  6
Number of observations :  478
Number of clusters :  239
```

These estimates match those from SAS. Also, notice that our scale parameter estimate is close to one, which is the theoretical value for a multinomial model.

G. Markov Chains: Transitional Modeling

Discrete-time Markov chains are discrete-time stochastic processes with a discrete state space. That means that the random variable (potentially) changes states at discrete time points (say, every hour), and the states come from a set of discrete (many times, also finite) possible states. So, if the state of the random variable at time t is represented by y_t, then the stochastic process is represented by $\{y_0, y_1, y_2, ...\}$.

This section deals with first-order Markov chains. Thus, for all *t*, the conditional distribution of y_{t+1} given $\{y_0, y_1, y_2, ..., y_t\}$ is identical to the distribution of y_{t+1} given only y_t. In words, this means that in order to predict the state at the next time point, we only need to consider the current state. Any states prior to the current state add nothing to predictability of the next state. This is sometimes called the Markov property.

The conditional probability, $P(Y_t = j \mid Y_{t-1} = i) = \pi_{j|i}(t)$, is called a one-step transition probability, because it is the probability of transition from state i at time $t - 1$ to state j at time t one time-step away. The sum of these probabilities over j is equal to 1 because at time t, the process must take on one of the possible states, even if it remains at its current state. Because of the Markov property, the joint distribution for a first-order Markov chain depends only on the one-step transition probabilities and on the marginal distribution for the initial state of the process.

Because association is only present between pairs of adjacent, consecutive states, a first-order Markov chain can be fit to a sample of realizations from the chain by fitting the loglinear model $(Y_0Y_1, Y_1Y_2, ..., Y_{T-1}Y_T)$ for T realizations. This model says that, for example, at any combination of states at the time points 2,…, T, the odds ratios describing the association between Y_0 and Y_1 are the same.

To illustrate how these models are fit, we fit several loglinear models to the Respiratory Illness data (Table 11.7 in Agresti). This was a longitudinal study of the effects of air pollution on respiratory illness in children, with yearly assessments done on children from age 9 to age 12. At each measurement, a child was classified according to the presence or absence of wheeze. Thus, the data represent a four-way contingency table, with two categories per dimension (presence, absence).

First, we read in the data.

```
table.11.7<-data.frame(expand.grid(Y12=c(1,0), Y11=c(1,0), Y10=c(1,0), Y9=c(1,0)),
        count=c(94,30,15,28,14,9,12,63,19,15,10,44,17,42,35,572))
```

A first-order Markov chain (MC) would assume an association between the response at 9 years and the response at 10 years, and the prediction at 11 years would be independent of the reponse at 9 years, given the response at 10 years. The prediction at 12 years would be independent of the response at 9 years and at 10 years, given the response at 11 years. That is, once we know the previous year, we don't need any other years to predict the response at a given year.

Here is a fit of the first-order MC model, using `loglm` from MASS. `loglm` is a front-end to `loglin`, which is much more complicated to use. The syntax is the same for R and S-PLUS. I have specified the arguments `param=TRUE` and `fit=TRUE` to get the parameter estimates and fitted values, respectively, as attributes.

```
library(MASS)
(fit.loglm1<-loglm(count~Y9*Y10 + Y10*Y11 + Y11*Y12, data=table.11.7, param=TRUE,
      fit=TRUE))

Statistics:
                     X^2 df P(> X^2)
Likelihood Ratio 122.9025  8        0
Pearson          129.5971  8        0
```

The LR statistic is very high at 122.9, on 8 degrees of freedom.

A *second*-order MC model assumes that the prediction at 12 years is independent of the response at 9 years, given *both* the responses at 10 years and at 11 years. This model is fit by

```
(fit.loglm2<-loglm(count~Y9*Y10*Y11 + Y10*Y11*Y12, data=table.11.7, param=TRUE,
fit=TRUE))

Statistics:
                     X^2 df     P(> X^2)
Likelihood Ratio 23.86324  4 8.507772e-05
Pearson          26.67938  4 2.307740e-05
```

Again, the LR statistic is high at 23.9, on 4 degrees of freedom.

The loglinear model with all pairwise associations gives the following.

```
(fit.loglm3<-loglm(count~ Y9*Y10 + Y11*Y12 + Y9*Y11 + Y10*Y12 + Y9*Y12 + Y10*Y11,
      data=table.11.7, param=TRUE, fit=TRUE))

Statistics:
                     X^2 df  P(> X^2)
Likelihood Ratio 1.458592  5 0.9177982
Pearson          1.446029  5 0.9192127
```

To get the parameter estimates, we call `fit.loglm3$param`. To get the conditional log odds ratios of Table 11.8 in Agresti, we do the same manipulating that we did in Chapter 8. For example, at each combination of 9 and 10-year responses, the odds of wheezing at 12 years old are

```
exp(fit.loglm3$param$Y11.Y12[1,1] + fit.loglm3$param$Y11.Y12[2,2] -
(fit.loglm3$param$Y11.Y12[1,2] + fit.loglm3$param$Y11.Y12[2,1]))

[1] 6.358584
```

times higher than the odds of wheezing at 11 years old. (The anti-log of this odds ratio gives the log odds ratio in Table 11.8).

The loglinear model with all pairwise associations, but constrained to have the one-year pair associations identical, and the greater-than-one-year associations identical fits well too. To fit this model, we create two new variables that represent, respectively, the one-year association and the greater-than-one-year association. These variables just count the number of such associations.

```
oneyear<-cbind(table.11.7$Y9*table.11.7$Y10, table.11.7$Y10*table.11.7$Y11,
      table.11.7$Y11*table.11.7$Y12)
gtoneyear<-cbind(table.11.7$Y9*table.11.7$Y11, table.11.7$Y9*table.11.7$Y12,
      table.11.7$Y10*table.11.7$Y12)
table.11.7$oneyear<-rowSums(oneyear)
table.11.7$gtoneyear<-rowSums(gtoneyear)
```

We will treat the two new variables as continuous instead of categorical. So, we can't use `loglm` to do the fitting (which is IPF). We will use `glm` instead. With `glm`, the factors in the model are not considered categorical.

```
options(contrasts=c("contr.treatment", "contr.poly")) # S-PLUS only
(fit.glm4<-glm(count~Y9 + Y10 + Y11 + Y12 + oneyear + gtoneyear, data=table.11.7,
family=poisson))

Coefficients:
(Intercept)           Y9          Y10          Y11          Y12      oneyear
      6.349       -2.227       -2.569       -2.687       -2.684        1.752

  gtoneyear
      1.036

Degrees of Freedom: 15 Total (i.e. Null);  9 Residual
Null Deviance:      2050
Residual Deviance: 2.269        AIC: 99.47
```

The parameter estimates here, conveniently, represent the conditional log odds ratios. The odds ratio above is now less than 6.0.

```
exp(coef(fit.glm4))["oneyear"]

 oneyear
5.766295
```

Transitional Models with Explanatory Variables

Transitional models can include explanatory variables. If the response is binary, the explanatory variables can be included in a logistic regression model regressing the current response on previous responses as well as the explanatory variables. The density for the *t*th sequential response (out of T responses) is then

$$f(y_t \mid y_1,...,y_{t-1};\mathbf{x}_t) = \frac{\exp\left[y_t(\alpha+\beta_1 y_1+...+\beta_{t-1}y_{t-1}+\boldsymbol{\beta}^T\mathbf{x}_t)\right]}{1+\exp(\alpha+\beta_1 y_1+...+\beta_{t-1}y_{t-1}+\boldsymbol{\beta}^T\mathbf{x}_t)},\ y_t=0,1$$

Thus, previous responses are treated as explanatory variables too. The product of the T terms for a given subject constitutes their contribution to the likelihood function. This model is called a regressive logistic model. This model can be fit using `glm`, treating multiple responses by an individual as separate data lines.

Next, I fit a first-order regressive logistic model to Table 11.9 in Agresti. This data set is from the same study of air pollution and health as Table 11.7. Children were evaluated annually on the presence of respiratory illness from age 7 to 10. A predictor is the presence/absence of maternal smoking (s = 0 for no smoking; s = 1 for regular smoking). The first-order regressive model uses the previous response as a predictor, along with maternal smoking and age. Thus, there are three responses per individual (for ages 8, 9, and 10). These are treated as separate observations in a `glm` fit. The process is the same in S-PLUS as in R.

The first object is a contingency table, like Table 11.9.

```
table.11.9.ref<-data.frame(expand.grid(Y10=c(0,1), Y9=c(0,1), Y8=c(0,1), Y7=c(0,1),
       mat.smoke=c(0,1)), count=c(237,10,15,4,16,2,7,3,24,3,3,2,6,2,5,11,118,
       6,8,2,11,1,6,4,7,3,3,1,4,2,4,7))
```

Next, we represent all 12 possible combinations of the values of the predictor variables.

```
table.11.9<-data.frame(expand.grid(previous=c(0,1), t=8:10, mat.smoke=c(0,1)))
```

Each row of `table.11.9` has a certain proportion of "presence" responses. The proportion comes from `table.11.9.ref`. Here is the count vector that represents the number of "presence" responses for each row.

```
count.yes<-c(
sum(table.11.9.ref[table.11.9.ref$Y8==1 & table.11.9.ref$mat.smoke==0 & table.11.9.ref$Y7==0,"count"]),
sum(table.11.9.ref[table.11.9.ref$Y8==1 & table.11.9.ref$mat.smoke==0 & table.11.9.ref$Y7==1,"count"]),
sum(table.11.9.ref[table.11.9.ref$Y9==1 & table.11.9.ref$mat.smoke==0 & table.11.9.ref$Y8==0,"count"]),
sum(table.11.9.ref[table.11.9.ref$Y9==1 & table.11.9.ref$mat.smoke==0 & table.11.9.ref$Y8==1,"count"]),
sum(table.11.9.ref[table.11.9.ref$Y10==1 & table.11.9.ref$mat.smoke==0 & table.11.9.ref$Y9==0,"count"]),
sum(table.11.9.ref[table.11.9.ref$Y10==1 & table.11.9.ref$mat.smoke==0 & table.11.9.ref$Y9==1,"count"]),
sum(table.11.9.ref[table.11.9.ref$Y8==1 & table.11.9.ref$mat.smoke==1 & table.11.9.ref$Y7==0,"count"]),
sum(table.11.9.ref[table.11.9.ref$Y8==1 & table.11.9.ref$mat.smoke==1 & table.11.9.ref$Y7==1,"count"]),
sum(table.11.9.ref[table.11.9.ref$Y9==1 & table.11.9.ref$mat.smoke==1 & table.11.9.ref$Y8==0,"count"]),
sum(table.11.9.ref[table.11.9.ref$Y9==1 & table.11.9.ref$mat.smoke==1 & table.11.9.ref$Y8==1,"count"]),
sum(table.11.9.ref[table.11.9.ref$Y10==1 & table.11.9.ref$mat.smoke==1 & table.11.9.ref$Y9==0,"count"]),
sum(table.11.9.ref[table.11.9.ref$Y10==1 & table.11.9.ref$mat.smoke==1 & table.11.9.ref$Y9==1,"count"])
)
```

Here is the same for "absence" responses.

```
count.no<-c(
sum(table.11.9.ref[table.11.9.ref$Y8==0 & table.11.9.ref$mat.smoke==0 & table.11.9.ref$Y7==0,"count"]),
sum(table.11.9.ref[table.11.9.ref$Y8==0 & table.11.9.ref$mat.smoke==0 & table.11.9.ref$Y7==1,"count"]),
sum(table.11.9.ref[table.11.9.ref$Y9==0 & table.11.9.ref$mat.smoke==0 & table.11.9.ref$Y8==0,"count"]),
sum(table.11.9.ref[table.11.9.ref$Y9==0 & table.11.9.ref$mat.smoke==0 & table.11.9.ref$Y8==1,"count"]),
sum(table.11.9.ref[table.11.9.ref$Y10==0 & table.11.9.ref$mat.smoke==0 & table.11.9.ref$Y9==0,"count"]),
sum(table.11.9.ref[table.11.9.ref$Y10==0 & table.11.9.ref$mat.smoke==0 & table.11.9.ref$Y9==1,"count"]),
sum(table.11.9.ref[table.11.9.ref$Y8==0 & table.11.9.ref$mat.smoke==1 & table.11.9.ref$Y7==0,"count"]),
sum(table.11.9.ref[table.11.9.ref$Y8==0 & table.11.9.ref$mat.smoke==1 & table.11.9.ref$Y7==1,"count"]),
sum(table.11.9.ref[table.11.9.ref$Y9==0 & table.11.9.ref$mat.smoke==1 & table.11.9.ref$Y8==0,"count"]),
sum(table.11.9.ref[table.11.9.ref$Y9==0 & table.11.9.ref$mat.smoke==1 & table.11.9.ref$Y8==1,"count"]),
sum(table.11.9.ref[table.11.9.ref$Y10==0 & table.11.9.ref$mat.smoke==1 & table.11.9.ref$Y9==0,"count"]),
sum(table.11.9.ref[table.11.9.ref$Y10==0 & table.11.9.ref$mat.smoke==1 & table.11.9.ref$Y9==1,"count"])
)
```

This is used to compute the proportions.

```
table.11.9$prop<-count.yes/(total<-count.yes+count.no)

fit.glm5<-glm(prop ~ previous + t + mat.smoke, data=table.11.9, family=binomial,
      weight=total)
summary(fit.glm5)


Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.29256    0.84603  -0.346   0.7295
previous     2.21107    0.15819  13.977   <2e-16 ***
t           -0.24281    0.09466  -2.565   0.0103 *
mat.smoke    0.29596    0.15634   1.893   0.0583 .
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 207.2212  on 11  degrees of freedom
Residual deviance:   3.1186  on  8  degrees of freedom
AIC: 64.392

Number of Fisher Scoring iterations: 3
```

The previous response has a strong effect. Conditional on the previous response, and t, the child's age, maternal smoking has a positive effect that is close to significant.

Recall that a LRT on any single coefficient is obtained by fitting another model without that coefficient, and subtracting residual deviances.

Chapter 12 – Random Effects: Generalized Linear Mixed Models for Categorical Responses

A. Summary of Chapter 12, Agresti

When observations occur in clusters, we can model the dependence within a cluster using (latent) random effects. With this model, there are as many random effects as observations. They take the same value for each observation in a cluster, but take different values across clusters. A cluster can be an observation itself, in which case the random effects will differ across observations. In a random effects model, the random effects apply to a sample of clusters instead of to a fixed set of clusters. That is, the cluster effects are considered randomly sampled from a distribution of such effects, and the actual realization of clusters will tend to differ across samples.

Random effects models are similar to the marginal models of Chapter 11 in that both models incorporate dependence among observations. However, with the marginal models the dependence is averaged over, and the marginal probability distribution is of more interest. With random effects models, we model the probability distribution within each cluster. Also, the (positive) correlation between observations within a cluster is assumed to have arose because the random effects vary across clusters.

Generalized linear mixed models (GLIMMs) extend generalized linear models (GLIMs) to incorporate random effects. Conditional on the random effect, a GLIMM resembles a GLIM. For observation t in cluster i, the linear predictor is

$$g(\mu_{it}) = \mathbf{x}_{it}^T\boldsymbol{\beta} + \mathbf{z}_{it}^T\mathbf{u}_i \qquad (12.1)$$

for link function g and conditional mean μ_{it}. The random effect vector, $\mathbf{u}_i$, is assumed to have a multivariate normal distribution $N(\mathbf{0}, \boldsymbol{\Sigma})$, where $\boldsymbol{\Sigma}$ describes the covariances among the elements of $\mathbf{u}_i$. Conditional on $\mathbf{u}_i$, the observations $\{y_{it}\}$ are independent over i and t. With the random effect appearing on the scale of the explanatory variables, subject-specific random effects can represent heterogeneity caused by omitting certain explanatory variables. Random effects can also represent random measurement error in the explanatory variables.

Because the $\mathbf{u}_i$ are random effects, their inclusion only increases the number of parameters in the model by the number of unique elements in their covariance matrix. On the other hand, if they were fixed effects, then the number of parameters would increase with the number of clusters. In that case, one can condition on the sufficient statistics for $\mathbf{u}_i$, making the resulting conditional likelihood independent of them. Maximizing this likelihood for $\boldsymbol{\beta}$ gives consistent estimates. Advantages of using a conditional likelihood include not having to specify a distribution for the random effects. Also, if the clusters are not randomly sampled (such as for a retrospective study), then conditional likelihood estimation is appropriate. However, conditioning removes the source of variability needed to estimate between-cluster effects (such as explanatory variable effects that are not repeated within clusters). Recall that conditional MLE only allows estimation of within-cluster effects (such a treatment effect within pairs in a matched-pairs design). Also, a conditional model will not allow prediction of the individual random effects. Finally, sufficient statistics for the $\mathbf{u}_i$ have to exist to use the approach.

The fixed effects parameters $\boldsymbol{\beta}$, both within-cluster and between-cluster, in a GLIMM have conditional interpretations, given the random effect. For example, in a cross-over drug trial, where each subject takes both the control drug and experimental drug, the drug effect (a within-subject effect), conditional on the subject's random effect is the coefficient that multplies the drug dummy variable. Also, in a multi-center drug trial, with a random center effect, where each center has patients who take the control drug as well as patients who take the experimental drug, the drug effect (a between-center effect), again conditional on a given center's random effect is the coefficient that multiples the drug dummy variable. This is only true when the control and experimental patient come from the same center. Thus, comparisons are done at a fixed value of the random effect. In contrast, effects in marginal models are

averaged over all clusters. When the link function is nonlinear, these effects are often smaller in magnitude than their analogous cluster-specific effects. The approximate relationship between the two parameters in a logistic-normal model has been shown to be that the marginal model coefficient is about $[1+0.6\sigma^2]^{-1/2}$ of the conditional coefficient, making it smaller in magnitude. Agresti's Figure 12.1 illustrates this attenuation.

With a multinomial outcome (ordinal or nominal), a random effects model is represented via the obvious extension. That is, one adds a random effect to the linear predictor in the same way as for a binomial outcome.

B. Logit GLIMM for Binary Matched Pairs

When g in (12.1) is the logit link, and μ_{it} is the conditional expectation of a Bernoulli random variable given u_i, then we have the logit GLIMM. Clustering occurs if the data are paired, so that each (matched) pair is a cluster. Suppose cluster i of n contains the matched pair (y_{i1}, y_{i2}), where y_{it} = 1 or 0, and given u_i,

$$\text{logit}[P(Y_{it}=1|u_i)] = \alpha + \beta x_t + u_i \qquad (12.2)$$

where $x_1 = 0$ and $x_2 = 1$, and $u_i \sim N(0, \sigma^2)$. Then, we have a logit GLIMM for binary matched pairs. The two elements of the pair are independent given u_i. So, their only dependence occurs through this random effect. If we compute the log odds ratio of responding 1 in each of the two pairs, we get β. That is,

$$\text{logit}[P(Y_{i2}=1|u_i)] - \text{logit}[P(Y_{i1}=1|u_i)] = \beta$$

within each cluster. Thus, β is called the cluster-specific log odds ratio. The model in (12.2) is called a *random intercept model* because the intercept of the linear predictor is a random effect instead of a fixed constant.

The marginal distribution of $Y_t = \sum_{i=1}^{n} y_{it}$ is binomial with index n and probability parameter $E\{\exp(\alpha+\beta x_t+U)/[1+\exp(\alpha+\beta x_t+U)]\}$, with expectation with respect to U a $N(0, \sigma^2)$ random variable. The model implies a non-negative correlation between Y_1 and Y_2, with greater association resulting from greater heterogeneity (larger σ^2). Greater heterogeneity means that all the u_i are spread apart, where there are both large positive and large negative values. Large positive values of u_i make it more likely that (y_{i1}, y_{i2}) are both 1. Large negative values of u_i make it more likely that (y_{i1}, y_{i2}) are both 0. This correspondence between the two elements of the match pair traverses to a correspondence between the two sums Y_1 and Y_2, leading to higher positive association.

The 2 x 2 population-averaged table represents the frequency distribution of (y_{i1}, y_{i2}). When the sample log odds ratio is non-negative, then the MLE of β can be shown to be equal to $\log(n_{21}/n_{12})$, which is the conditional MLE, using conditional maximum likelihood estimation as discussed in Agresti's chapter 10. A sample log odds ratio that is negative means that the estimate of the variance, σ^2, is zero, implying independence. Then, the MLE of β is identical to that from the marginal model for matched pairs.

Agresti uses the Prime Minister Ratings data to illustrate a random intercept model. The matched pairs are the clusters. The data are available in table form from Agresti's website. I copied the data and placed it into a text file called `primeminister.txt`. Then, I read it into R/S-PLUS.

```
(table.12.1<-read.table("primeminister.txt",col.names=c("case", "occasion",
"response", "count")))
```

```
  case occasion response count
1    1        0        1   794
2    1        1        1   794
3    2        0        1   150
4    2        1        0   150
5    3        0        0    86
6    3        1        1    86
7    4        0        0   570
8    4        1        0   570
```

Now, we should make a new data frame that contains the correct number of cases, 3200. This can be accomplished by repeating the 2nd and 3rd columns of each pair of rows, `count` number of times. Then, we ensure that the case variable counts sequently in doubles. Thus,

```
temp1<-cbind(case=rep(1:794,each=2),table.12.1[rep(1:2,794),2:3])
temp2<-cbind(case=rep(795:(794+150),each=2),table.12.1[rep(3:4,150),2:3])
temp3<-cbind(case=rep(945:(944+86),each=2),table.12.1[rep(5:6,86),2:3])
temp4<-cbind(case=rep(1031:(1030+570),each=2),table.12.1[rep(7:8,570),2:3])

table.12.1a<-rbind(temp1,temp2,temp3,temp4)
```

Now, I turn `case` into a factor.

```
table.12.1a$case<-factor(table.12.1a$case)
```

There are several functions in R from different packages that fit generalized linear mixed models, including random intercept models. One function that is common to both R and S-PLUS is `glmmPQL` in the MASS library. This function fits the model using the penalized quasi-likelihood approximation to maximum likelihood, which is not always accurate for binary data. Here, it is a little off. In the formula, we specify that there is a random intercept (`random=~1`) and that it corresponds to each `case` value.

```
library(MASS)
fit.glmmPQL<-glmmPQL(response~occasion, random=~1 | case , family=binomial,
data=table.12.1a)

summary(fit.glmmPQL)       # R output

Linear mixed-effects model fit by maximum likelihood
 Data: table.12.1a
       AIC      BIC    logLik
  17249.61 17273.89 -8620.803

Random effects:
 Formula: ~1 | case
        (Intercept)  Residual
StdDev:    3.263896 0.5414447

Variance function:
 Structure: fixed weights
 Formula: ~invwt
Fixed effects: response ~ occasion
                Value  Std.Error   DF   t-value p-value
(Intercept)  0.8499475 0.10012885 1599  8.488538       0
occasion    -0.5688280 0.07295128 1599 -7.797368       0
 Correlation:
```

```
         (Intr)
occasion -0.376

Standardized Within-Group Residuals:
       Min         Q1        Med         Q3        Max
-2.1613715 -0.4746219  0.3097340  0.4117406  2.0973320

Number of Observations: 3200
Number of Groups: 1600
```

The MLE for the (cluster-specific) log odds ratio of approval (beta) is –0.569, with SE = 0.073, so that the estimated odds of approval at the second survey are exp(–0.569) = 0.57 times that at the first survey. The estimate of the standard deviation of the random effects is 3.26.

Other R functions that fit GLIMMs use other types of approximations (namely, maximization of the numerically integrated likelihood -- integrated over the normal distrubtion of the random effects). The repeated library has a function called `glmm`.

```
library(repeated)
fit.glmm<-glmm(response~occasion , nest=case, family=binomial, data=table.12.1a)

summary(fit.glmm)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   1.3190     0.1111   11.87  < 2e-16 ***
occasion     -0.5540     0.1332   -4.16 3.18e-05 ***
sd            4.6830     0.1759   26.62  < 2e-16 ***
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 4373.2  on 3199  degrees of freedom
Residual deviance: 3501.1  on 3197  degrees of freedom
AIC: 3507.1

Number of Fisher Scoring iterations: 20
```

In the call, the argument `nest` specifies the cluster variable.

In the output, the MLE for the (cluster-specific) log odds ratio of approval (beta) is –0.554, with SE = 0.133, so that the estimated odds of approval at the second survey are exp(–0.554) = 0.57 times that at the first survey. The estimate of the standard deviation of the random effects is 4.68. These numbers are closer to what SAS NLMIXED gets for Agresti. (In fact, it is the same fitting algorithm as NLMIXED).

The R package glmmML also fits GLIMMs by maximizing the numerically integrated likelihood

```
library(glmmML)
(fit.glmmML<-glmmML(response~occasion, cluster=table.12.1a$case, family=binomial,
data=table.12.1a))



              coef se(coef)      z Pr(>|z|)
(Intercept)  1.3404   0.2495  5.372 7.78e-08
occasion    -0.5563   0.1353 -4.113 3.91e-05

Standard deviation in mixing distribution:  5.138
Std. Error:                                 0.4339

Residual deviance: 3502 on 3197 degrees of freedom      AIC: 3508
```

In the call, the argument `cluster` specifies the cluster variable. Apparently, the variable is not taken from the data frame, and must be referenced (e.g., by using the $).

In the output, the MLE for the (cluster-specific) log odds ratio of approval (beta) is –0.556, with SE = 0.135, so that the estimated odds of approval at the second survey are exp(–0.556) = 0.57 times that at the first survey. The estimate of the standard deviation of the random effects is 5.14. These numbers are closest to what SAS NLMIXED gives. Using the approximate relationship given in Section A, this implies a marginal effect of about –0.135.

Another R function called `glmm` in package GLMMGibbs fits GLIMMs via MCMC sampling.

The extension of (12.2) to more than 2 observations in each cluster is sometimes called the Rasch Model. In this case the model would look like

$$\text{logit}\left[P(Y_{it}=1|u_i)\right]=\beta_t+u_i \tag{12.3}$$

for t = 1,…,T. Note that the log odds ratio of responding 1 at t = 1 versus t = 2 , for a given subject i is the difference, $\beta_1-\beta_2$. This is not the same as the population-averaged log odds ratio, which would be $\text{logit}\left[P(Y_1=1)\right]-\text{logit}\left[P(Y_2=1)\right]$, and is the subject of marginal models.

C. Examples of Random Effects Models for Binary Data

1. Small Area Estimation of Binomial Proportions

Small area estimation models are random effects models. These models treat each area as a cluster with its own random effect coming from a common distribution of the random effects. The model estimates each area's mean by borrowing information from other areas. The borrowing takes place through the parameters of the common distribution. The resulting estimates of the area means are shrunk toward the population mean. The amount of shrinkage is controlled by the size of the estimated variance of the random effects distribution, and the size of the area. A lower variance and lower size result in less shrinkage.

Let Y_{it} be tth binary response for the ith area, t = 1,…,T_i. Conditional on the random effect, u_i ~ $N(0,\sigma^2)$,

$$\text{logit}\left[P(Y_{it}=1|u_i)\right]=\alpha+u_i \tag{12.4}$$

so that the estimate of $\pi_i=P(Y_{it}=1|u_i)$ is a function of the predicted random effect, $\hat{u}_i$, which is estimated by its marginal posterior mean.

Table 12.2 in Agresti gives data on simulated voting proportions in the 1996 presidential election for each of the 50 states. The states are the areas.

After reading in the data (which are available on Agresti's web site for the text), we fit a random intercept model using `glmmPQL` from the MASS library (available in both S-PLUS and R). The response is the observed proportion, with "case" weights as the number of observations per state.

```
table.12.2<-read.table("vote.txt",col.names=c("y","n"))
table.12.2$state<-1:nrow(table.12.2)

library(MASS)
```

```
fit.glmmPQL<-glmmPQL(y/n~1, random=~1 | state , weights=n, family=binomial,
data=table.12.2)
summary(fit.glmmPQL)

Linear mixed-effects model fit by maximum likelihood
 Data: table.12.2
       AIC      BIC    logLik
  84.34555 90.14103 -39.17278

Random effects:
 Formula: ~1 | case
        (Intercept)  Residual
StdDev:   0.3030006 0.9451296

Variance function:
 Structure: fixed weights
 Formula: ~invwt
Fixed effects: y/n ~ 1
                Value Std.Error DF  t-value p-value
(Intercept) 0.1582088 0.0677359 51 2.335671  0.0235

Standardized Within-Group Residuals:
         Min           Q1          Med           Q3          Max
-2.216914105 -0.480966832 -0.009083338  0.313103383  1.794107257

Number of Observations: 51
Number of Groups: 51
```

The estimate of the standard deviation of the random effects is 0.303, and is relatively small. So, the state estimates will show a lot of shrinkage to the population mean. We can get the state estimates using the fitted function

```
exp(fitted(fit.glmmPQL))/(1+exp(fitted(fit.glmmPQL)))
```

To use the `glmm` function in the repeated package in R, we have to construct the response so that it is a two-column matrix of the number of successes and failures.

```
library(repeated)
fit.glmm<-glmm(cbind(y,n-y)~1 , nest=state, family=binomial, data=table.12.2)
summary(fit.glmm)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.16615    0.04553   3.650 0.000263 ***
sd           0.28593    0.04441   6.439 1.21e-10 ***
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 95.671  on 50  degrees of freedom
Residual deviance: 77.284  on 49  degrees of freedom
AIC: 262.2

Number of Fisher Scoring iterations: 10

Normal mixing variance: 0.08175857
```

The estimates are similar to those from `glmmPQL`, but not identical.

2. Modeling Repeated Binary Responses

We can add covariates to model (12.3), and then consider the multiple questions or items for an individual as part of a single cluster. Agresti illustrates fitting this model using the abortion data in Table 10.13. In that data set, each subject (cluster) answered three yes/no questions. Subjects were classified by sex. A random effects logistic model for the response assumes a positive correlation between responses from the three questions by a single individual.
We start with the previous data frame called `table.10.13` (see chapter 10, above).

```
table.10.13
```

```
  gender question1 question2 question3 count
1      M         Y         Y         Y   342
2      F         Y         Y         Y   440
3      M         N         Y         Y     6
4      F         N         Y         Y    14
5      M         Y         N         Y    11
6      F         Y         N         Y    14
7      M         N         N         Y    19

etc
```

We reshape the data so that it is in “long” format instead of “wide” format. In R, one can use the function `reshape` or `stack` for this; in S-PLUS, one can use the function `menuStackColumns`. `reshape` is definitely easier to use than `menuStackColumns`.

```
menuStackColumns(target=table.10.13a,source=table.10.13,group.col.name="question",targ
    et.col.spec="response", source.col.spec=c("question1","question2","question3"),
    rep.source.col.spec=c("gender","count"),rep.target.col.spec=list(gender="gender",co
    unt="count"),show.p=F)
#R: table.10.13a<-reshape(table.10.13, varying=c("question1","question2",
    "question3"), drop="symm", direction="long", v.names="response",
    timevar="question")
```

In the next command, I reverse the levels of the factor, question, so that when I fit the model, the last coefficient is set to zero instead of the first, for identifiability constraints.

```
table.10.13a$question<-factor(table.10.13a$question, levels=rev(c("question1",
    "question2","question3")))
#R: table.10.13a$question<-factor(table.10.13a$question,levels=3:1)
```

Now, I create a new table that replicates the respective rows according to the counts in each cell. I also create an id variable that indexes the subject, and I make response into a 0/1-valued variable instead of a factor, so that we can fit a binomial model.

```
table.10.13b<-table.10.13a[rep(1:nrow(table.10.13a),table.10.13a$count),
    c("response","gender","question")]
table.10.13b$id<-factor(rep(1:1850,3))
table.10.13b$response<-c(unclass(table.10.13b$response)-1)
```

Now, I fit the random effects model, with `id` as the cluster. First, I use the function `glmmPQL` in the MASS library in both S-PLUS and R, then I use the function `glmmML`, in the R package, glmmML.

The function `glmmPQL` uses penalized quasi-likelihood estimation (maximizing a LaPlace approximation to the likelihood function), which can be inaccurate for binary data (see Agresti p. 524).

```
options(contrasts=c("contr.treatment", "contr.poly")) # S-PLUS only

library(MASS)
fit.glmmPQL<-glmmPQL(response~gender+question, random=~1|id, family=binomial,
    data=table.10.13b)
summary(fit.glmmPQL)
```

```
Linear mixed-effects model fit by maximum likelihood
 Data: table.10.13b
       AIC      BIC    logLik
  32091.19 32130.92 -16039.59

Random effects:
 Formula:  ~ 1 | id
        (Intercept)  Residual
StdDev:    4.308437 0.4507951

Variance function:
 Structure: fixed weights
 Formula:  ~ invwt
Fixed effects: response ~ gender + question
                      Value Std.Error   DF   t-value p-value
      (Intercept) -0.5108888 0.1710876 3698  -2.98612  0.0028
           gender  0.0064619 0.2215622 1848   0.02917  0.9767
questionquestion2  0.3067315 0.0722624 3698   4.24469  <.0001
questionquestion1  0.8677367 0.0727694 3698  11.92447  <.0001

Standardized Within-Group Residuals:
       Min         Q1        Med        Q3      Max
 -3.997665 -0.2546468 -0.2182912 0.2840273 3.835893

Number of Observations: 5550
Number of Groups: 1850
```

These estimates and their standard errors are somewhat off from SAS's NLMIXED estimates, but match those of SAS's GLIMMIX macro (see Agresti's comment on p. 524). Using `glmmML` gives estimates and SE's more similar to those from SAS. Both functions maximize the integrated likelihood.

```
library(glmmML)
(fit.glmmML<-glmmML(response~gender+question, cluster=table.10.13b$id,
family=binomial, data=table.10.13b))
```

```
               coef se(coef)       z Pr(>|z|)
(Intercept) -0.06137   0.1803 -0.3404 7.34e-01
genderF     -0.03620   0.1986 -0.1823 8.55e-01
question2    0.30681   0.1606  1.9102 5.61e-02
question1    0.83473   0.1601  5.2122 1.87e-07

Standard deviation in mixing distribution:  6.348
Std. Error:                                 0.1981

Residual deviance: 4571 on 5545 degrees of freedom      AIC: 4581
```

Thus, regardless of gender, the estimated odds of supporting legalized abortion for item 1 equal exp(0.83) = 2.3 times the estimated odds for item 3. The estimated standard deviation of the random effects distribution is 6.3, here (somewhat smaller than NLMIXED's 8.6, with a much smaller standard error). A higher standard deviation indicates greater heterogeneity among subjects, but high association within responses from a given subject.

The methods from Chapter 11 can be used to fit a marginal model to this data set (GEE and MLE).

We can obtain the fitted cell frequencies by numerically integrating over the distribution of the random effects, and then multiplying the result by the number of observations in the stratum (here, male or female). So, the probability that we estimate is the joint probability of a particular set of responses averaged over individuals. This is

$$P(Y_{i1}=y_{i1},Y_{i2}=y_{i2},Y_{i3}=y_{i3})=\int P(Y_{i1}=y_{i1},Y_{i2}=y_{i2},Y_{i3}=y_{i3}\,|\,u_i)\,f(u_i)du_i$$
$$=\int\prod_{t=1}^{3}P(Y_{it}=y_{it}\,|\,u_i)\,f(u_i)du_i$$

The probabilities $P(Y_{it}=y_{it}\,|\,u_i)$ are defined in equation (12.10) in Agresti (p. 504), and the density of u is normal. So, we numerically integrate. The function in R/S-PLUS to integrate is called `integrate`, and is very easy to use, theoretically. Another function that works correctly for this problem is in the R package `rmutil`, and is called `int`. For either, we first define the function to be numerically integrated. Here, I compute the probability of a positive response for each of the three questions, for a female respondent. Then, to get the expected frequency for that cell of the contingency table, we multiply the estimated probability by 1037, the number of female observations.

First, we get the estimates of the parameters from the fit. Then, we set a variable called `female` to 1, to indicate a female.

```
beta.coef<-
    c(fit.glmmPQL$coefficients$fixed[c("questionquestion1","questionquestion2")],0)
#R: beta.coef<-c(fit.glmmML$coef[c("question1","question2")],0)
female<-1

invlogit<-function (x) {exp(x)/(1+exp(x))}
```

The function, called `f`, is different depending on whether we use the fitted result from `glmmPQL` or `glmmML`.

```
# glmmPQL
f<-function(u){
    prod(invlogit(u+beta.coef+fit.glmmPQL$coefficients$fixed["(Intercept)"]+
    fit.glmmPQL$coefficients$fixed["gender"]*female))*dnorm(u,0,exp(attr(fit.glmmPQL$ap
    Var,"Pars")["reStruct.id"])
)
}

# glmmML (R only)
f<-function(u){
    prod(invlogit(u+fit.glmmML$coef["(Intercept)"]+ beta.coef +
    fit.glmmML$coef["genderF"]*female))*dnorm(u,0,fit.glmmML$sigma)
}
```

The expected frequency results from calling int or integrate with this function, and multiplying by 1037. It gives about 439.4, which is close to the observed frequency of 440. Agresti gets 436.5.

```
# R only
library(rmutil)
1037*int(f)
[1] 439.3701


result<-integrate(f,lower=-Inf,upper=Inf)  # this gives an incorrect answer.
1037*result$integral
```

Agresti also illustrates repeated binary responses via a longitudinal study, the mental depression data from Chapter 11 (Table 11.2). Thus, the three repeated measurements occur at distinct calendar times. Let y_{it} be observation t for subject i (1 = normal; 0 = abnormal). Then, conditional on the random effect for subject i

$$\text{logit}\left[P(Y_{it}=1\,|\,u_i)\right]=\alpha+\beta_1 s+\beta_2 d+\beta_3 t+\beta_4 dt+u_i$$

where s is the severity of diagnosis (1 = severe, 0 = mild) and d is the drug treatment (1 = new, 0 = standard). The coefficients have subject-specific interpretations.

The data are already in the object `table.11.2` (see above). Here, we fit a random intercept model using the R function `glmmML` in the glmmML package, the R function glmm in the repeated package, and the function `glmmPQL` in the MASS library.

First, the `glmmML` R function, because it gives the "correct" MLEs.

```
library(glmmML)
(fit.glmmML<-glmmML(outcome~diagnose+treat*time, cluster=table.11.2$case,
family=binomial, data=table.11.2))
```

```
                coef se(coef)       z Pr(>|z|)
(Intercept) -0.02795   0.1641 -0.1703 8.65e-01
diagnose    -1.31521   0.1546 -8.5046 0.00e+00
treat       -0.05970   0.2225 -0.2684 7.88e-01
time         0.48284   0.1160  4.1638 3.13e-05
treat:time   1.01842   0.1924  5.2930 1.20e-07

Standard deviation in mixing distribution:  0.06581
Std. Error:                                 1.242

Residual deviance: 1162 on 1014 degrees of freedom      AIC: 1174
```

As Agresti notes, the subject-specific parameter estimates are very close to the GEE and ML marginal estimates. The reason he gives is that the estimate of the standard deviation of the random effects distribution is close to zero (0.07), implying little heterogeneity among subjects. Thus, the derived pairwise correlation among observations on the same subject is not high (see equation (12.6) in Agresti).

Now, we fit the same model using `glmm` in the `repeated` R package.

```
library(repeated)
fit.glmm<-glmm(outcome~diagnose+treat*time, nest=case, family=binomial,
data=table.11.2, points=10)
```

```
Coefficients:
(Intercept)     diagnose        treat         time           sd   treat:time
   -0.02741     -1.33422     -0.06115      0.48917      0.26182      1.03267

Degrees of Freedom: 1019 Total (i.e. Null);  1014 Residual
Null Deviance:      1412
Residual Deviance: 1162         AIC: 1174
Normal mixing variance: 0.06854993
```

Notice that it estimates the *variance* of the random effects distribution to be 0.07 instead of the standard deviation.

```
summary(fit.glmm)
```

```
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.02741    0.16514  -0.166 0.868150
diagnose    -1.33422    0.14783  -9.025  < 2e-16 ***
treat       -0.06115    0.22378  -0.273 0.784652
time         0.48917    0.11560   4.231 2.32e-05 ***
sd           0.26182    0.07263   3.605 0.000313 ***
treat:time   1.03267    0.19017   5.430 5.62e-08 ***
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1411.9  on 1019  degrees of freedom
Residual deviance: 1161.7  on 1014  degrees of freedom
AIC: 1173.7
```

The function glmmPQL in the MASS library does the same.

```
library(MASS)
fit.glmmPQL<-glmmPQL(outcome~diagnose+treat*time, random=~1|case, family=binomial,
data=table.11.2)
summary(fit.glmmPQL)

Linear mixed-effects model fit by maximum likelihood
 Data: table.11.2
       AIC      BIC    logLik
  4611.546 4646.039 -2298.773

Random effects:
 Formula: ~1 | case
        (Intercept)  Residual
StdDev:   0.3151588 0.9719163

Variance function:
 Structure: fixed weights
 Formula: ~invwt
Fixed effects: outcome ~ diagnose + treat * time
                Value Std.Error  DF   t-value p-value
(Intercept) -0.0263963 0.1627507 678 -0.162189  0.8712
diagnose    -1.3170725 0.1469893 337 -8.960329  0.0000
treat       -0.0614527 0.2194690 337 -0.280006  0.7796
time         0.4825385 0.1118404 678  4.314528  0.0000
treat:time   1.0209475 0.1843255 678  5.538828  0.0000

Number of Observations: 1020
Number of Groups: 340
```

For predictions, we can use the function `int` again, from the `rmutil` R package, and the results from `fit.glmmML`. Here, I compute the expected number of cases in the cell (N, N, N) for severe diagnosis (1), New drug (1).

```
invlogit<-function (x) {exp(x)/(1+exp(x))}
diagnosis<-1
drug<-1
beta.coef<-(fit.glmmML$coef["time"] + drug*fit.glmmML$coef["treat:time"])*0:2

f<-function(u){
      prod(invlogit(u+fit.glmmML$coef["(Intercept)"]+
      fit.glmmML$coef["diagnose"]*diagnosis+beta.coef))*dnorm(u,0,fit.glmmML$sigma)
}

library(rmutil)
340*int(f)
[1] 8.583693
```

Compare this to the observed 7 cases.

3. Modeling Heterogeneity among Multicenter Clinical Trials

When a clinical trial is run across several centers, the treatment effects within the centers can be regarded as random effects, sampled from a population of such center effects. Inference is then extended to the population distribution. In addition, because the center effects borrow information from each other via their parent population distribution, the treatment effect within each center is estimated with more precision than if it had been estimated independently of the remaining centers.

Agresti uses the clinical trial data from Chapter 6 (Table 6.9, repeated in Table 12.5), comparing a drug to a control, to illustrate fitting a logistic-normal model to multicenter clinical trial data. For a subject in center i using treatment t (1 = drug, 2 = control), let $y_{it} = 1$ denote success. A general model includes a treatment-by-center interaction:

$$\text{logit}\left[P(Y_{i1}=1|u_i,b_i)\right]=\alpha+(\beta+b_i)/2+u_i$$

$$\text{logit}\left[P(Y_{i2}=1|u_i,b_i)\right]=\alpha-(\beta+b_i)/2+u_i$$

where $u_i \sim N(0,\sigma_a^2)$ and $b_i \sim N(0,\sigma_b^2)$, all independent of one another. With this model, $\beta+b_i$ is the center-specific log odds ratio for the ith center, and β is the expected center-specific log odds ratio. Thus, we have random center effects and random treatment-by-center effects. When σ_b^2 = 0, the log odds ratio between treatment and response is constant over centers.

Recall that we read these data into S-PLUS/R as a data frame in Section E of Chapter 6. I make a few changes to this data frame so that we have the coding used by Agresti.

```
table.6.9$TreatC<-ifelse(table.6.9$Treatment=="Drug",.5,-.5)
table.6.9$RespC<-ifelse(table.6.9$Response=="Success",1,0)
```

Now, I expand the data frame so that each row is repeated `Freq` times.

```
table.6.9a<-table.6.9[rep(1:nrow(table.6.9),table.6.9$Freq), c("RespC","TreatC",
"Center")]
```

To fit the random intercepts model as well as the random treatment-by-center interaction, I use `glmmPQL` in the MASS library, because it is the only function that allows specifying more than one cluster variable. It is available in both S-PLUS and R. I show R output.

First, I fit the model with random intercepts in Center.

```
library(MASS)
fit.glmmPQL.int<-
glmmPQL(RespC~TreatC, random=~1|Center, family=binomial, data=table.6.9a)
summary(fit.glmmPQL.int)

Linear mixed-effects model fit by maximum likelihood
 Data: table.6.9a
       AIC      BIC    logLik
  1271.560 1285.998 -631.7799

Random effects:
 Formula: ~1 | Center
        (Intercept)  Residual
StdDev:    1.323061 0.9652609

Variance function:
 Structure: fixed weights
 Formula: ~invwt
Fixed effects: RespC ~ TreatC
               Value Std.Error  DF   t-value p-value
(Intercept) -0.7799349 0.5019781 264 -1.553723  0.1214
TreatC       0.7210237 0.2864029 264  2.517515  0.0124
 Correlation:
       (Intr)
TreatC -0.012

Standardized Within-Group Residuals:
       Min         Q1        Med         Q3        Max
-2.0633991 -0.6235402 -0.3736916  0.7461471  3.5571301
```

```
Number of Observations: 273
Number of Groups: 8
```

The estimate of 0.721 (SE = 0.286) is a little off from Agresti's MLE of 0.739 (SE = 0.300). Note Agresti's discussion of penalized quasi-likelihood in Section 12.6.4.

Now, I fit the treatment-by-center interaction as well.

```
fit.glmmPQL.ia<-
glmmPQL(RespC~TreatC, random=~TreatC|Center, family=binomial, data=table.6.9a)
summary(fit.glmmPQL.ia)

Linear mixed-effects model fit by maximum likelihood
 Data: table.6.9a
       AIC      BIC    logLik
  1278.671 1300.328 -633.3354

Random effects:
 Formula: ~TreatC | Center
 Structure: General positive-definite, Log-Cholesky parametrization
            StdDev    Corr
(Intercept) 1.3489277 (Intr)
TreatC      0.3083281 -0.867
Residual    0.9627373

Variance function:
 Structure: fixed weights
 Formula: ~invwt
Fixed effects: RespC ~ TreatC
                Value Std.Error  DF   t-value p-value
(Intercept) -0.8130185 0.5116942 264 -1.588876  0.1133
TreatC       0.8084582 0.3120634 264  2.590686  0.0101
 Correlation:
       (Intr)
TreatC -0.332

Standardized Within-Group Residuals:
       Min         Q1        Med         Q3        Max
-1.8942090 -0.6317714 -0.3874070  0.7163946  3.9731951

Number of Observations: 273
Number of Groups: 8
```

The treatment effect is estimated to be 0.81 (SE = 0.312), slightly different than the MLE of 0.746 (SE = 0.325. Note, again, that `glmmPQL` apparently estimates the *variance* of the center-specific treatment coefficients to be around 0.15 (well, here 0.09), not the standard deviation. This variance represents variability in the log odds ratios across centers. The fact that the standard deviation is estimated to be quite small indicates that the log odds ratios are probably similar and that the standard errors of the treatment association parameter are similar across models.

Predicted odds ratios for either model are easy to obtain. The `predict` or `fitted` method for `glmmPQL` (really, `lme`) gives the predicted logits. A simple transformation gives the predicted odds ratios. More direct is to extract the estimated coefficients of the logit model using `coef`, and multiplying these by the design matrix. Below, I do this for each treatment separately, get the inverse logits, and take the ratio of the two results.

Here are the coefficient estimates for the interaction model. These include both the fixed and random parts for each center.

```
coef(fit.glmmPQL.ia)
```

```
  (Intercept)    TreatC
a  -0.8998236 0.7692297
b   0.9721856 0.4584772
c   0.1236859 0.6716927
d  -2.1502939 1.0664178
e  -1.4833405 0.9778767
f  -2.4923268 1.1419009
g  -1.5186057 0.9455523
h   0.9443708 0.4365184
```

Now, I get the predicted odds ratios.

```
invlogit<-function (x) {exp(x)/(1+exp(x))}

odds.treat<-invlogit(as.matrix(coef(fit.glmmPQL.ia))%*%c(1,.5))
odds.control<-invlogit(as.matrix(coef(fit.glmmPQL.ia))%*%c(1,-.5))

odds.treat/odds.control
```

```
       [,1]
  a 1.725004
  b 1.134493
  c 1.370664
  d 2.589480
  e 2.210855
  f 2.860352
  g 2.164892
  h 1.130364
```

Unfortunately, these predicted odds ratios are quite a bit off from those of SAS NLMIXED (compare to Table 12.5 in Agresti). It is unclear whether this is due to the use of PQL as an estimation technique or whether there is a mistake in the computation above.

4. Capture-Recapture Modeling to Predict Population Size

The standard capture-recapture experiment to estimate the size of a population uses two or more sources. Each source is a sample from the population. After the first sample is taken, the elements are recorded and returned to the population. After the second sample is taken, the elements are recorded and returned. The second sample might recapture some elements that were sampled first. For T samples from the population, a 2^T contingency table records how many elements had each of the various capture patterns (e.g., captured at occasion 1, not captured at occasion 2, etc). However, the cell representing non-capture at any occasion is an unknown. Knowing this cell value would provide us with the population size.

The probability of capture at a given occasion can be heterogeneous across sources. One model to represent heterogeneous capture probabilities is a random effects logit model, with random subject effects. Thus, for subject i, let $\mathbf{y}_i^T = (y_{i1},...,y_{iT})$, where $y_{it} = 1$ denotes capture in sample t, and $y_{it} = 0$ denotes non-capture, and $i = 1,\ldots, N$, with N unknown. Then, a logistic-normal model could be

$$\text{logit}\left[P(Y_{it} = 1 \mid u_i)\right] = u_i + \beta_t$$

where u_i are independent $N(0,\sigma^2)$. The larger the value of β_t, the larger the probability of capture at occasion t. The larger is σ, the greater the heterogeneity among capture probabilities.

To get the likelihood, we can integrate the random effect from the probability mass function conditional on the random effect,

$$p(\mathbf{y}_i \mid u_i) = \prod_t \left[\frac{\exp(\mathbf{x}_{it}^T \boldsymbol{\beta} + u_i)}{1+\exp(\mathbf{x}_{it}^T \boldsymbol{\beta} + u_i)} \right]^{y_{it}} \left[\frac{1}{1+\exp(\mathbf{x}_{it}^T \boldsymbol{\beta} + u_i)} \right]^{1-y_{it}}$$

then treat the $\boldsymbol{y}_i$'s as multinomial draws, with probabilities given by $p(\mathbf{y}_i) = \int_u p(\mathbf{y}_i \mid u_i)\, p(u_i \mid \sigma^2)\, du$, and index N. Maximizing this likelihood gives the MLE of N, and the other parameters. This is straightforward to do in S-PLUS or R, using a function like `optim`. However, to match the estimate obtained by Agresti, I will also illustrate a conditional approach that Agresti used in Coull and Agresti, 1999 (see references in Agresti).

First, we should enter the data from Table 12.6. I do this in several steps so that I can the data in different forms throughout this section.

Here, I enter the 0/1 patterns, separately from the counts.

```
temp<-matrix(c(
c(0,0,1,0,0,0),
c(0,1,0,0,0,0),
c(0,1,1,0,0,0),
c(1,0,0,0,0,0),
c(1,0,1,0,0,0),
c(1,1,0,0,0,0),
c(1,1,1,0,0,0),

…… etc., see code file

c(0,0,0,1,1,1),
c(0,0,1,1,1,1),
c(0,1,0,1,1,1),
c(0,1,1,1,1,1),
c(1,0,0,1,1,1),
c(1,0,1,1,1,1),
c(1,1,0,1,1,1),
c(1,1,1,1,1,1)), byrow=T,nc=6)

counts<-c(3,6,0,5,1,0,0,3,2,3,0,0,1,0,0,4,2,3,1,0,1,0,0,1,0,0,0,0,0,0,0,4,
1,1,1,2,0,2,0,4,0,3,0,1,0,2,0,2,0,1,0,1,0,1,0,1,1,1,0,0,0,1,2)
```

Now, I put everything into a data frame, and add names.

```
table.12.6<-data.frame(temp,counts)
names(table.12.6)<-c(paste("Survey",c(3:1,6:4),sep=""),"counts")
```

Next, I repeat the `temp` rows, `counts` times, to get the right number of rows.

```
table.12.6a<-table.12.6[rep(1:nrow(table.12.6),table.12.6$counts),1:6]
row.names(table.12.6a)<-1:nrow(table.12.6a)
```

Next, I change the shape of the data frame so that it is "long" instead of "wide". In R, the function `reshape` is very easy to use. In S-PLUS, one can use `menuStackColumns`. I present `reshape` primarily.

```
#S-PLUS: table.12.6a$id<-1:nrow(table.12.6a)
table.12.6b<-reshape(table.12.6a,direction="long",
varying=list(names(table.12.6a)), timevar="Survey",v.names="Capture")
```

```
#S-PLUS: menuStackColumns(source=table.12.6a,target=table.12.6b,
    source.col.spec=names(table.12.6a)[1:6],group.col.name="Survey",
    target.col.spec="Capture",rep.source.col.spec="id",show.p=F)
#S-PLUS: names(table.12.6b)<-c("Capture", "id", "Survey" )
```

Then, I make `Survey` a factor, as we will be using it as such in fitting models.

```
table.12.6b$Survey<-factor(table.12.6b$Survey)
```

Now, I fit a model to just the known data. The resulting parameter estimates are used as starting values for the maximization.

```
library(MASS)
fit.glmmPQL<-glmmPQL(Capture~Survey-1, random=~1|id, family=binomial,
data=table.12.6b)
beta.coef<-fit.glmmPQL$coef$fixed
```

Now, I create the negative log likelihood for the unknown parameters, including the unknown size, N. The function is called `func2`, and takes as a single argument, a vector of the unknown parameters. I use the log of N instead of N. Also, the function as written can only be used in R because it uses `int`. To use in S-PLUS, try to replace the line `int(f)` with `integrate(f,lower=-Inf,upper=Inf)`.

```
temp2<-rbind(rep(0,6),temp)   # add the all-zero row
invlogit.i<-function (x,i) {exp(i*x)/(1+exp(x))}

func2<-function(y) {

    counts<-c(exp(y[8]),3,6,0,5,1,0,0,3,2,3,0,0,1,0,0,4,2,3,1,0,1,0,0,1,
      0,0,0,0,0,0,0,4,1,1,1,2,0,2,0,4,0,3,0,1,0,2,0,2,0,1,0,1,0,1,0,1,1,
      1,0,0,0,1,2)

    require(rmutil)     # R only

    probs<-apply(temp2,1,function(x){
        i<-x
        f<-function(u){
            prod(invlogit.i(y[1:6]+u,i))*dnorm(u,0,exp((y[7])))
        }
        int(f)
    })

#S-PLUS:   probs<-apply(temp2,1,function(x,y){
#        f<-function(u,x,y){
#            prod(invlogit.i(y[1:6]+u,x))*dnorm(u,0,exp((y[7])))
#        }
#        integrate(f,lower=-Inf,upper=Inf,x=x,y=y)$integral
#    },y=y)

    -lgamma(sum(counts)+1) +sum(lgamma(counts+1)) - sum(counts*log(probs))
}
```

Now, I get the MLEs.

```
result<-optim(par=c(beta.coef,log(fit.glmmPQL$sigma),log(18)),func2,
control=list(trace=6,REPORT=1),method="BFGS")
result.values<-result$par
```

```
c(result.values[1:6],exp(result.values[7:8]))

   Survey1    Survey2    Survey3    Survey4    Survey5    Survey6
-1.4378836 -0.9094129 -1.7539897 -0.6751367 -1.2273151 -1.0325512  0.9075391 20.9831241
```

The last value (plus 68) is N, and the penultimate value is σ. These are close to those obtained by Agresti, 24+68 and 0.97, respectively.

Next, I compute the profile-likelihood confidence interval, using the function `plkhci`, which is only available in the R package, `Bhat`. To obtain the interval in S-PLUS, one could use a brute force method by inputting different values of the missing cell count, and obtaining the log likelihood.

Recall that to use the function `plkhci`, we create a list with the labels, starting values, and lower and upper bounds. Then, we input this list as an argument, followed by the negative log-likelihood function, and the label of the parameter being profiled. A confidence interval is returned. Here, it is about (74, 141), similar to that obtained by Agresti (75, 154).

```
library(Bhat)

x <- list(label=c("beta1","beta2","beta3","beta4","beta5","beta6",
"lsig","lN"), est=result.values, low=c(rep(-4,6),-1,-10), upp=c(rep(4,6), 2,
10))

ans<-plkhci(x,func2,"lN",prob=0.95)
exp(ans)+68
[1]  73.39961 141.07961
```

Another way to estimate N is to use a conditional estimation. Here, we estimate the binomial likelihood of observing n = 68 successes (captures) in N trials, when the probability of success is $1-\pi_{0..0}(\hat{\boldsymbol{\theta}}_C)$, where $\hat{\boldsymbol{\theta}}_C$ maximizes the conditional likelihood

$$L_1(\boldsymbol{\theta}; n_{observed} \mid n) \propto \pi'_{1\cdots1}(\boldsymbol{\theta})^{n_{1\cdots1}} \cdots \pi'_{0\cdots1}(\boldsymbol{\theta})^{n_{0\cdots1}}$$

where $\pi'_{\mathbf{i}} = \pi_{\mathbf{i}}(\boldsymbol{\theta}) \Big/ \left\{ \sum_{\mathbf{i} \in observed} \pi_{\mathbf{i}}(\boldsymbol{\theta}) \right\}$, $\pi_{\mathbf{i}}(\boldsymbol{\theta}) = \int_u \pi_{\mathbf{i}|u}\ p(u \mid \sigma^2)\, du$, and **i** ranges only over the observed patterns. The estimate of N is then $n \Big/ \left\{ 1-\pi_{0\cdots0}(\hat{\boldsymbol{\theta}}_C) \right\}$.

First, I write the function for the negative conditional log likelihood, as a function of the unknown parameters, observed counts, observed number of captures, and observed patterns.

```
func<-function(Beta,counts,n,matrix.i) {

    require(rmutil)      # R only

    probs<-apply(matrix.i,1,function(x){
        i<-x
        f<-function(u){
            prod(invlogit.i(Beta[1:6]+u,i))*dnorm(u,0,exp(Beta[7]))
        }
        int(f)     # S-PLUS: integrate(f, lower=-Inf, upper=Inf)
    })
    -sum(counts*log(probs/sum(probs)))
```

```
}
```

Now, we minimize the negative log likelihood and obtain the MLEs of the parameters. The MLE of N is obtained using the expression given above.

```
junk<-optim(par=c(beta.coef,log(fit.glmmPQL$sigma)),fn=func,
control=list(trace=2), method="BFGS", counts=counts, n=68,matrix.i=temp)

beta.coef.000<-junk$par[1:6]
sigma.000<-exp(junk$par[7])

invlogit<-function (x) {exp(x)/(1+exp(x))}

prob.000000<-function(u){
    prod(1-invlogit(beta.coef.000+u))*dnorm(u,0,sigma.000)
}

library(rmutil)
68/(1-int(prob.000000))
[1] 92.00398
```

Thus, the total estimated size of the population is 92 snowshoe hares.

Other programs and functions for capture-recapture include WISP, which works under R 2.0, and is available from http://www.ruwpa.st-and.ac.uk/estimating.abundance/WiSP/ . Also, the `vegan` package for R 2.0 contains functions for estimating species abundance. The function `specpool` can be used for this problem.

D. Random Effects Models for Multinomial Data

No new principles are introduced with the extension of random effects models for multinomial data. As with binary data, one adds one or more random effects to the linear predictor for the cumulative logit or adjacent-categories logit model, for example. However, in terms of model fitting, it is harder to do the estimation because there are more probabilities to estimate. And, there are much fewer options available in R or S-PLUS. In fact, the only available package option as of the date of this writing for fitting multinomial random effects models is the R function `logitord`, in the `repeated` package, which claims to fit random effects ordinal regression models, with dropout. However, the function does not work well for the examples in Agresti's text, and indeed, even for the examples provided in the function's own help file. There it gives NaN's for log likelihood values, and SEs that are too large to be believable.

Thus, to fit these models in S, we are apparently forced to either link our own compiled code to the software, or use a brute force method, by directly maximizing the numerically integrated likelihood. Here, I illustrate the latter, but it can be painfully slow especially if starting values are not close to the maximizer. Incidentally, there is always a third option, which is to use the `system` function to invoke a windows batch file for an executable program that fits multinomial random effects models (e.g., the downloadable MIXOR program). This option is relatively easy to use by reading the help file for `system`, but as this manual discusses the use of S, I will not illustrate that method.

Agresti uses the insomnia data to fit a cumulative logit model with random intercept. The form is a proportional odds model, and so we can obtain starting values by first fitting a proportional odds model without random effects. We can use the function `polr` in the MASS library (in both R and S-PLUS). After reading in the data, we negate the variables to be used in the model because `polr` fits the "negatives" of their coefficients (see Section C of Chapter 7 of this manual).

```
table.11.4<-read.table("c:/program files/r/rw1080/cda/insomnia.txt", header=T)
table.11.4$outcome<-factor(table.11.4$outcome) # polr wants a response factor
```

```
table.11.4$treat<--table.11.4$treat
table.11.4$time<--table.11.4$time

library(MASS)
fit<-polr(outcome~treat*time, data=table.11.4)
```

We will use the `optim` function to do the maximization. For this we want the data in "wide" format, with one column for each occasion response.

```
table.11.4a<-reshape(table.11.4, direction="wide", v.names = "outcome", timevar =
"time", idvar = "case",drop="count")
# make sure all numeric columns, and exclude case
table.11.4a<-apply(table.11.4a,2,as.numeric)[,-1]

# S-PLUS
# junk<-structure(.Data=
# menuUnstackColumns(source=table.11.4, source.col.spec=c("outcome"), group=c("time"),
    show.p=F),names=c("outcome.0","outcome.1"))
# table.11.4a<-data.frame(treat=c(rep(-1,238/2),rep(0,(478-238)/2)),junk)
```

Now, we define the objective function to be the log of the product of the category probabilities (see Agresti's equation (7.6))

```
invlogit<-function (x) {exp(x)/(1+exp(x))}

func<-function(Beta,matrix.i) { # three alphas, three betas, one log sigma

    require(rmutil) # R only

    probs<-apply(matrix.i,1,function(x){
    # S-PLUS: probs<-apply(matrix.i,1,function(x,Beta){

        f<-function(u){
        # S-PLUS: f<-function(u,Beta){

        treat<-x[1]

        prob1.0<-prob1.1<-prob2.0<-prob2.1<-prob3.0<-prob3.1<-prob4.0<-prob4.1<-1

        if(x[2]==1) prob1.0<-invlogit(Beta[1] + Beta[5]*treat + u)
              else if(x[2]==2) prob2.0<-invlogit(Beta[2] + Beta[5]*treat + u) -
              invlogit(Beta[1] + Beta[5]*treat + u)
              else if(x[2]==3) prob3.0<-invlogit(Beta[3] + Beta[5]*treat + u) -
              invlogit(Beta[2] + Beta[5]*treat + u)
              else if(x[2]==4) prob4.0<-1-invlogit(Beta[3] + Beta[5]*treat + u)
       if(x[3]==1) prob1.1<-invlogit(Beta[1] + Beta[4] + Beta[5]*treat + Beta[6]*treat
       + u)
              else if(x[3]==2) prob2.1<-invlogit(Beta[2] + Beta[4] + Beta[5]*treat +
              Beta[6]*treat + u) - invlogit(Beta[1] + Beta[4] + Beta[5]*treat +
              Beta[6]*treat + u)
              else if(x[3]==3) prob3.1<-invlogit(Beta[3] + Beta[4] + Beta[5]*treat +
              Beta[6]*treat + u) - invlogit(Beta[2] + Beta[4] + Beta[5]*treat +
              Beta[6]*treat + u)
              else if(x[3]==4) prob4.1<-1-invlogit(Beta[3] + Beta[4] + Beta[5]*treat +
              Beta[6]*treat + u)

prob1.0*prob1.1*prob2.0*prob2.1*prob3.0*prob3.1*prob4.0*prob4.1*dnorm(u,0,exp(Beta[7])
)}
        int(f)
        # S-PLUS: integrate(f,-Inf,Inf,Beta=Beta)
    })
    # S-PLUS: },Beta=Beta)
```

```
    -sum(log(probs))
}
```

Now, we use the `optim` function. Here, I start the optimization at the estimates from `polr`, but to have it converge in any reasonable amount of time, you have to start very close to the maximizer, which appears to be `c(-3.4858986, -1.4830128, 0.5606742,1.602,.058,1.081,log(1.9))`

```
optim(par=c(fit$zeta,fit$coefficients[c("time","treat","treat:time")],.5), fn=func,
control=list(trace=2,REPORT=1), method="BFGS", matrix.i= as.matrix(table.11.4a))
```

```
$par
[1] -3.48824875 -1.48407698  0.56211694  1.60158887  0.05754634  1.08140528
[7]  0.64423760

$value
[1] 592.9728

$counts
function gradient
      20        4

$convergence
[1] 0

$message
NULL
```

Fitting adjacent-categories logit model follows the same idea, by defining the likelihood as a product of probabilities within each category.

E. Multivariate Random Effects Models for Binary Data

One form of multivariate random effects is the presence of both a random intercept and a random slope for a regression that "varies" across clusters, like individuals. Another type is nested random effects, where for example, one can have random effects for schools and random effects for students within schools. Finally, one can have a model with more than one response variable, where each response variable has an associated random subject effect, say.

1. Bivariate Binary Response

Agresti gives an example of a bivariate binary response where schoolboys were interviewed twice, several months apart, and asked about their self-perceived membership in the leading crowd (yes or no) and asked about whether they sometimes felt they needed to go against their principles in order to fit in to that crowd (yes or no). Thus, the two binary responses are called Membership and Attitude. There are subject random effects for each variable.

Following Agresti, for subject i, let y_{itv} be the response at interview time t on variable v. Then, we use the logit model

$$\text{logit}\left[P(Y_{itv}=1|u_{iv})\right]=\beta_{tv}+u_{iv}$$

where $(u_{iAtt}, u_{iMem}) \sim N(\mathbf{0},\boldsymbol{\Sigma})$. Thus, the correlation between Membership and Attitude is considered to be possibly nonzero.

The data in Table 12.8 in Agresti are available on his website. Here I read in the data set after copying it and saving it as a text file called "crowd.txt".

```
table.12.8<-read.table("c:/program files/r/rw2000/cda/crowd.txt", header=T)
```

There is no library function for either R or S-PLUS (yet) that automatically fits multivariate logit models with random effects via maximum likelihood or any other method. There are several ways we might do the fitting. One straightforward method is to maximize the integrated likelihood, where integration is over the distribution of the bivariate random effects (of course, we use numerical integration). Multivariate Gaussian quadrature can be accomplished either via the `adapt` function in the `adapt` library or using a simple 2D numerical integration function, called `GL.integrate.2D`, from Diego Kuonen, and modified slightly below.

```
GL.integrate.2D <-function(fct, low, upp, order=10,...)
{
YW.list<-as.list(1:2)
for(i in 1:2) {
    name.YW<-paste("GL.YW", abs(low[i]), abs(upp[i]), order,
        sep=".")
    if(!exists(name.YW)){
        assign(name.YW, GL.YW(order,
            xrange=c(low[i],upp[i])),
            pos=1, immediate=T)  # for S-PLUS, use where=1 instead of pos=1
        }
    YW.list[[i]]<-get(name.YW)
    }

point<-expand.grid(YW.list[[1]][,1], YW.list[[2]][,1])

#fcteval<-outer2(YW.list[[1]][,1], YW.list[[2]][,1], fct,...)      # original
fcteval<-apply(point,1,fct,...)  # added by LAT
fcteval<-matrix(fcteval,nr=order,nc=order,byrow=F)    # added by LAT
sum(YW.list[[1]][,2]*apply(YW.list[[2]][,2]*fcteval,2,sum))
}
```

The points and the weights for the Gaussian quadrature are computed using the function `GL.YW`, also from Kuonen, and included in the code files for this manual.

We can perform the optimization using the function `optim`. We have seven parameters: the four coefficients describing the time x variable combinations and the three parameters in the covariance matrix. I parameterize the covariance matrix using the correlation coefficient instead of the covariance in order to constrain the matrix to be positive definite. I also use the conditional representation of the pdf.

Here I define the function that will be maximized (called func below). As before, all the parameters are contained within one vector, called Beta here. I integrate the bivariate normal distribution from –10 to 10, which is approximate. The following is the function as called within R. The function for S-PLUS is within the S-PLUS code file for this manual.

```
invlogit<-function (x) {exp(x)/(1+exp(x))}

func<-function(Beta,matrix.i) {

    require(adapt)          # R only
    require(mvtnorm)        # R only

    sigma11<-exp(Beta[5])
    sigma22<-exp(Beta[6])

    probs<-apply(matrix.i,1,function(x){

        f<-function(u){

        mem1<-x[1]
        att1<-x[2]
        mem2<-x[3]
```

```
        att2<-x[4]

        prob.mem1<-invlogit(Beta[1]+ u[1])
        prob.mem2<-invlogit(Beta[2]+ u[1])

        prob.att1<-invlogit(Beta[3]+ u[2])
        prob.att2<-invlogit(Beta[4]+ u[2])

        log.prob<-mem1*log(prob.mem1) + (1-mem1)*log(1-prob.mem1) +
         mem2*log(prob.mem2) + (1-mem2)*log(1-prob.mem2) + att1*log(prob.att1) +
        (1-att1)*log(1-prob.att1) + att2*log(prob.att2) + (1-att2)*log(1-prob.att2)

       exp(log.prob)*exp(dnorm(u[1], mean=0, sd=sigma11,log=T))* exp(dnorm(u[2],
       mean=u[1]*Beta[7]*sigma22/sigma11, sd=sigma22*sqrt(1-Beta[7]^2),log=T))
        }

        GL.integrate.2D(fct=f, low=rep(-10,2), upp=rep(10,2))

    })

    -sum(matrix.i[,5]*log(probs))
}
```

For the call to `optim`, I start at the beta coefficients that are fitted by PROC NLMIXED in SAS. I also use the variance values given by Agresti.

```
res<-optim(par=c(-1.105,-.74,.21,.39,log(3.1),log(1.5),.3),
fn=func,control=list(trace=6,REPORT=1), method="L-BFGS-B", hessian=T, lower=c(rep(-
Inf,6),-.99), upper=c(rep(Inf,6),.99), matrix.i=as.matrix(table.12.8))

res

$par
[1] -1.1399494 -0.7738794  0.3423025  0.5568294  1.0738703  0.5967430  0.2884974

$value
[1] 8544.888

$counts
function gradient
      15       15

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

Unfortunately, starting at values far from the solution appear to converge to values far from those above. It is possible that other types of optimization methods, such as an extension of the “EM algorithm” would perform better, but I have not tried.

Since we asked for the approximate second derivative matrix, we can extract it to try to approximate the standard errors.

```
se<-sqrt(diag(solve(res$hessian)))
names(se)<-c("beta1m","beta2m","beta1a","beta2a","log(Sm)","log(Sa)","rho")
se
    beta1m     beta2m     beta1a     beta2a    log(Sm)    log(Sa)        rho
0.08210803 0.08047563 0.06241528 0.06142253 0.03845709 0.02673202 0.02877469
```

Another option is to take advantage of an existing function for generalized linear mixed models, such as `glmmPQL` from library `MASS`, and do some tricks to emulate a multivariate response with multivariate random effects. A couple of emails from the R-news newsgroup deal with fitting these models, and one references a write-up in the multilevel website (http://multilevel.ioe.ac.uk/softrev/reviewmixed.pdf) which explains how to fit a bivariate normal response (not repeated measures) with random effects, using `lme`. I applied this method to `glmmPQL` (which uses `lme`) for a bivariate binary response.

Here, I set up the original data set so that have one column for "response", and add two additional indicator columns that indicate from which variable the response comes. I also add a time indicator variable that indicates whether the response is measured at time 1 or time 2.

```
data1<-table.12.8[rep(1:nrow(table.12.8),table.12.8$count),]
data1<-cbind(id=1:nrow(data1),data1[,1:4])
data1<-reshape(data1, direction="long", varying= list(c("mem1","att1","mem2","att2")),
v.names="response", timevar="occasion")
# S-PLUS: menuStackColumns(target=data1.2, source=data1, source.col.spec=
   c("mem1","att1","mem2","att2"), target.col.spec="response",
rep.source.col.spec="id",rep.target.col.spec ="id", group.col.p=F, show.p=F)

data2<-cbind(data1$response, data1$id,rep(c(1,0,1,0), each=nrow(data1)/4),
rep(c(0,1,0,1), each=nrow(data1)/4), rep(c(1,1,0,0),each=nrow(data1)/4))
#S-PLUS:  data2<-cbind(data1.2$response, data1.2$id,rep(c(1,0,1,0),
   each=nrow(data1.2)/4), rep(c(0,1,0,1), each=nrow(data1.2)/4), rep(c(1,1,0,0),
each=nrow(data1.2)/4))

data2<-as.data.frame(data2)
names(data2)<-c("response","id","var1","var2","time")
```

Here are the first 10 rows of `data2`

```
data2[1:10,]
```

	response	id	var1	var2	time
1	1	1	1	0	1
2	1	2	1	0	1
3	1	3	1	0	1
4	1	4	1	0	1
5	1	5	1	0	1
6	1	6	1	0	1
7	1	7	1	0	1
8	1	8	1	0	1
9	1	9	1	0	1
10	1	10	1	0	1

So, these first 10 observations are for variable 1 (membership) at time 1 (first interview). In the call to `glmmPQL`, I remove the intercept in both the fixed and random specifications. I do so in the random specification in order to get different random intercepts for each variable. I also do not include a single fixed term for "time". Unfortunately, the algorithm only runs for four iterations on this data set before hitting a NaN in the likelihood at iteration 5.

```
# S-PLUS: library(MASS)
temp<-glmmPQL(response~-1 + var1 + var2 + var1:time + var2:time, data=data2, family=
binomial, random=~-1+var1+var2|id)    # only runs for 4 iterations in R version
summary(temp)

Linear mixed-effects model fit by maximum likelihood
 Data: data2
       AIC     BIC    logLik
  63897.46 63957.6 -31940.73

Random effects:
 Formula:  ~ -1 + var1 + var2 | id
```

```
 Structure: General positive-definite
           StdDev  Corr
    var1 2.034878 var1
    var2 1.578995 0.265
Residual 0.771258

Variance function:
 Structure: fixed weights
 Formula:  ~ invwt
Fixed effects: response ~ -1 + var1 + var2 + var1:time + var2:time
               Value  Std.Error    DF   t-value p-value
     var1 -0.4992500 0.05009042 10191 -9.966977  <.0001
     var2  0.3585770 0.04156703 10191  8.626476  <.0001
var1:time -0.2922856 0.05012462 10191 -5.831177  <.0001
var2:time -0.1699343 0.04392112 10191 -3.869079  0.0001

Standardized Within-Group Residuals:
       Min        Q1        Med        Q3     Max
 -1.531963 -0.586571 -0.3687342 0.5954922 1.60645

Number of Observations: 13592
Number of Groups: 3398
```

Nonetheless, the coefficient estimates are –0.4992 – 0.2923 = – 0.7915 for membership at first interview (compare to –1.105 with SAS), –0.4992 for membership at second interview (compare to –0.7398 with SAS), 0.3586 – 0.1699 = 0.1887 for attitude at first interview (compared to 0.2139 with SAS), and 0.3586 for attitude at second interview (compared to 0.3897 with SAS). Standard errors are close to those from SAS for the attitude coefficients, but are almost half those from SAS for the membership coefficients.

The estimate of the standard deviation of the random effect for membership is 2.03 (compared to 2.78 from SAS) and for attitude is 1.58 (compared to 1.48 from SAS). The estimated correlation between the random effects is 0.265 (compared to 0.33 from SAS). The "residual" estimate is an estimate of the scale parameter for the binomial. Here it is less than the theoretical value of 1.0.

It has been mentioned in the literature that penalized quasi-likelihood can give standard errors that are too low when the random-effects variances are large. In fact for this example, the SAS estimate (which uses ML) of the random-effect variance for membership is 7.73, which probably qualifies as large if it is close to the actual variance. This might explain the discrepancies in standard errors.

2. Clustered Ordinal Outcomes

One can also fit continuation-ratio logit models with random effects by considering the set of frequencies of the cumulative ordinal responses *within a cluster* as a sequence of independent binomial variates. Following Agresti, for observation t in cluster i, let $\omega_{ij} = P(Y_{it} = j \mid Y_{it} \geq j, u_{ij})$. Let n_{ij} be the number of subjects in cluster i making response j, and $n_i = \sum_{j=1}^{I} n_{ij}$ for I responses. Then, the n_{ij}s can be considered independent binomial($n_i - \sum_{h<j} n_{ih}, \omega_{ij}$) variates ($j$ = 1,…, I – 1).

To illustrate fitting this model with S software, I use the same data set as Agresti's Table 12.9. This study examines the developmental effects of ethylene glycol by administering four doses to pregnant rodents within litters. There are three possible ordered outcomes for the fetus: dead, malformed, and normal. The continuation-ratio model fits the probability of death, as well as the probability of malformation given alive (i.e., malformed or normal).

Following Agresti, for litter i in dose group d, let $\text{logit}(\varpi_{i(d)1})$ be the continuation-ratio logit for the probability of death and $\text{logit}(\varpi_{i(d)2})$ the continuation-ratio logit for the conditional probability of

malformation given survival. Also, let x_d be the dosage for group d. In the model, we use litter-specific random effects: $\mathbf{u}_{i(d)} = (u_{i(d)1}, u_{i(d)2})$ sampled from a $N(\mathbf{0}, \mathbf{\Sigma}_d)$. The bivariate random effect allows differing amounts of dispersion for the two probabilities. We also fit separate fixed dose effects for the two logits, $\text{logit}(\varpi_{i(d)j}) = u_{i(d)j} + \alpha_j + \beta_j x_d$, where j is the index for the logit.

Here I read in the data from Table 12.9, which I have placed in a text file called toxic.txt. This file has column headings: litter, dose, dead, malformed, and normal. The last 3 columns come directly from the numbers in Table 12.9 (I will have the data posted on the main website for this manual).

```
table.12.9<-read.table("c:/toxic.txt", header=T)
doses<-c(0,.75,1.5,3)
```

The next set of functions define the objective function to be maximized, that is the likelihood. I illustrate a likelihood that assumes no correlation between the two random effects. It is much faster to estimate, and is the likelihood estimated by Agresti. One can use either the `adapt` function (in the R library `adapt`) to do the numerical integration in the likelihood, or one can use `GL.integrate.2D`, from Diego Kuonen (a modified version is given above, as well as in the posted script file). `GL.integrate.2D` is faster. The S-PLUS script uses `GL.integrate.2D` (see S-PLUS script for details). I integrate from –5 to 5, which appears to be adequate.

```
invlogit<-function (x) {exp(x)/(1+exp(x))}

func.common.sig<-function(Beta,matrix.i) {

    require(adapt)  # with R only
    require(mvtnorm)   # with R only

    probs<-apply(matrix.i,1,function(x){
        f<-function(u){

        sigma.dead<-exp(Beta[5])
        sigma.mal<-exp(Beta[6])

        dose<-doses[x[2]]
        dead<-x[3]
        mal<-x[4]
        normal<-x[5]
        n<-sum(x[3:5])

        prob.dead<-invlogit(Beta[1]*dose + Beta[2] + u[1])
        prob.mal.given.alive<-invlogit(Beta[3]*dose + Beta[4] + u[2])

        log.prob<-dead*log(prob.dead)  + mal*log(prob.mal.given.alive) +
        (normal)*log(1-prob.mal.given.alive)  + (n-dead)*log(1-prob.dead)

        exp(log.prob)*exp(dnorm(u[1], mean=0, sd=sigma.dead,log=T))*
            exp(dnorm(u[2],mean=u[1]*Beta[7]*sigma.mal/sigma.dead,
sd=sigma.mal*sqrt(1-Beta[7]^2), log=T))
        }
#        adapt(ndim=2,lower=rep(-5,2), upper=rep(5,2), eps=.001,functn=f)$value
        GL.integrate.2D(fct=f, low=rep(-5,2), upp=rep(5,2))

    })

    -sum(log(probs))
}
```

Now, I estimate some starting values, using a continuation-ratio model without random effects. We can do this with `vglm` from library VGAM in R (discussed in Section 7F of ths manual). Note that I use `aggregate` to eliminate the litter column.

```
library(VGAM)
table.12.9a<-aggregate(table.12.9,by=list(dose=table.12.9$dose),FUN=sum)
# type rm(logit) if the command below gives an error about logit function
fit.vglm<-vglm(cbind( dead, malformed,normal )~doses,family=sratio(link ="logit",
parallel = F), data=table.12.9a)
```

Now, I estimate the random effects model using `optim` (maximum likelihood). I permute the starting values to correspond to the ordering used in the `func.common.sig` function. I also use logs of the standard deviations.

```
res<-optim(par=c(coef(fit.vglm)[c(3,1,4,2)],log(1),log(1),0),
fn=func.common.sig, method="L-BFGS-B", hessian=T, lower=c(rep(-Inf,6),-.99),
upper=c(rep(Inf,6),.99), matrix.i=as.matrix(table.12.9))

iterations 18
function evaluations 22
norm of the final projected gradient 0.00255419
final function value 465.634

X = 0.0589047 -4.44258 1.70543 -4.3003 -0.0415667 0.442632 0.0381847
F = 465.634
final  value 465.634452
converged
```

The MLEs are

```
c(res$par[1:4],sigma1=exp(res$par[5]),sigma2=exp(res$par[6]),rho=res$par[7])

      doses:1 (Intercept):1       doses:2 (Intercept):2        sigma1
   0.05890473   -4.44257580    1.70543299   -4.30029525    0.95928532

       sigma2           rho
   1.55679960    0.03818472
```

These are mostly quite similar to those from Coull and Agresti (cited in Agresti). However, their MLE for sigma1 is about 0.55.

You can also get the approximate standard errors if `hessian=T` was specified in the `optim` function. I report the standard errors on the *logs* of the standard deviations.

```
sqrt(diag(solve(res$hessian)))

      doses:1 (Intercept):1       doses:2 (Intercept):2
    0.2289091     0.4135260     0.2428424     0.4738901     0.2187893

    0.1355963     0.3843808
```

Because the standard error for the dose effect on death (0.23) is greater in magnitude than its estimate (0.06), we can conclude that there is no evidence of a dose effect on probability of death. However, according to the MLEs, the estimated odds of malformation given survival multiply by exp(1.71) = 5.5 for every additional g/kg of ethylene glycol. Based on the estimates of the standard deviations of the random effects, the litter effect is also stronger on malformation given survival than it is on death.

In the R script, I give a version of the objective function where distinct covariance matrices are used (it is called “func”). This takes quite a while to estimate.

Chapter 13 – Other Mixture Models for Categorical Data

A. Summary of Chapter 13, Agresti

The models used to describe heterogeneity or overdispersion in Chapter 12 are mixture models, with Normal mixing distributions (distribution for the random effects). Chapter 13 describes mixture models with non-Normal mixing distributions, including nonparametric mixing distributions. Latent class models use a categorical mixing distribution. Nonparametric random effect models use unspecified distributions on a finite set of mass points. You can use this type of distribution if you don't want to make any assumptions about the form of the random effects distribution. However, because the locations and probabilities of the mass points are parameters that have to be estimated, I am not sure non-parametric is not technically a misnomer. Certainly, these are not parameters of interest, though.

Beta-binomial models, negative binomial regression (as gamma mixtures of poisons), and Poisson regression with random effects are all discussed as parametric alternatives to binary GLMMS and Poisson models.

B. Latent Class Models

Latent class models use a categorical mixing distribution, with "latent" categories. Given knowledge of this latent categorical variable, the observed categorical response variables are independent. Thus, for a randomly selected individual with observed responses, $(y_1,...,y_T)$ on T random variables, given that we know which category of a latent random variable Z the individual falls in,

$$P(Y_1 = y_1,...,Y_T = y_T \mid Z = z) = P(Y_1 = y_1 \mid Z = z)\cdots P(Y_T = y_T \mid Z = z)$$

Summing over the distribution of the latent variable, we get the joint probability of the observed responses.

$$\begin{aligned} P(Y_1 = y_1,...,Y_T = y_T) &= \sum_{z=1}^{q} P(Y_1 = y_1,...,Y_T = y_T \mid Z = z)P(Z = z) \\ &= \sum_{z=1}^{q}\left[\prod_{t=1}^{T} P(Y_t = y_t \mid Z = z)\right]P(Z = z) \\ &= \pi_{y_1,...,y_T} \end{aligned} \qquad (13.1)$$

Theoretically, one can imagine an underlying variable or class that describes a set of observed responses. Agresti notes that latent class analysis is the analog of factor analysis for category response variables. But, even if such a factor can't be imagined, one can still fit a latent class model to a set of observed categorical responses that appear to be overdispersed with respect to a particular parametric distribution.

If each response has I categories, then the model assumes a multinomial distribution over the I^Tcells, with probabilities in (13.1) and corresponding counts $\{n_{y_1...y_T}\}$. The EM algorithm can be used to obtain MLEs of the probabilities. The algorithm iterates between calculating pseudocounts for the unobserved table, which are $n_{y_1...y_T}$ for each value of Z. Then, the pseudocounts are treated as data in fitting a loglinear

model $(Y_1Z, Y_2Z, ..., Y_TZ)$, in order to get estimates of expected counts $\mu_{y_1,...,y_T,Z}$. Then, updated pseudocounts at the (s+1) iteration are obtained using the formula

$$n^{(s+1)}_{y_1,...,y_T,Z} = n_{y_1,...,y_T,Z} \frac{\mu^{(s)}_{y_1,...,y_T,Z}}{\sum_{k=1}^{q} \mu^{(s)}_{y_1,...,y_T,k}}$$

One can use the converged estimates of fitted probabilities to calculate the MLEs of the latent class model parameters: the conditional probabilities $P(Y_t = y_t \mid Z = z)$ (there are $qT(I-1)$ of these) and the q marginal probabilities $P(Z = z)$.

1. Latent Class Models for Reader Agreement

Agresti uses the pathologist data set (with additional pathologists) as an example of a latent class model fit to ratings data. Each pathologist rated 118 slides on the precence or absence of carcinoma in the uterine cervix. Two latent classes are proposed: one for those slides whose true rating is positive, and one for those slides whose true rating is negative.

Since the data are not available on Agresti's website, I create a matrix of the patterns of ratings given by the seven pathologists, along with their frequencies, then expand the patterns by the frequency corresponding to each pattern. To generate the patterns, I use the function `combn` in the R library `combinat`. This function produces all possible samples of a given size from a finite set. Here, the finite set is (1,…,7) for the 7 pathologists. Each sample tells which pathologists rated the slide a 1 (cancer).

```
library(combinat)
table.13.1<-matrix(0,nc=7,nr=2^7) # start with all zeroes (negatives)

i<-1
sapply(1:6, function(y) {
    nrows<-ncol(cols<-combn(1:7,y))
    sapply(1:nrows, function(x) {
        table.13.1[i,cols[,x]]<<-1 # place a 1 in these columns, for ith row
        i<<-i+1   # increment the row each time
        })
})

table.13.1[2^7,]<-rep(1,7)  # add the all 1 row
```

Now, I add the frequencies.

```
table.13.1.full<-cbind(table.13.1,counts<-c(2,6,0,0,2,0,0,
2,0,0,0,0,0,0,0,4,0,1,0,0,0,0,0, 0,0,0, 0,0, 0, 0,0,2,0,1, 0,0,0,0, 0,0,0,0,0,0,
0,0,0,0, 0,0,0, 0,5,0, 0,0,0, 0,0,0, 0,0,0, 0,0,0,0,0, 0,0,1, 0,7,0,
0,0,0,0,1, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,13, 0,0,2,0,1,
0,0,0,0,0, 0,0,0,0,0, 0,0,10,0,5,3, 0,0,34,16))
```

Remove the patterns with zero frequencies, and expand the patterns by the frequencies.

```
table.13.1<-table.13.1.full[table.13.1.full[,8]!=0,]
table.13.1<-table.13.1[rep(1:nrow(table.13.1), table.13.1[,8]),]
```

Function lca

There are several R functions to fit latent class models. First, I use the `lca` function in the R library `e1071`. I enter as arguments the result of a tabulation of the patterns, using `countpattern`, and the number of classes (`k`) and iterations.

```
library(e1071)
summary(res.lca<-lca(countpattern(table.13.1[,1:7]),k=2,niter=15))


LCA-Result
----------

Datapoints: 118
Classes:    2

Goodness of fit statistics:

Number of parameters, estimated model: 15
Number of parameters, saturated model: 127
Log-Likelihood, estimated model:       -317.2574
Log-Likelihood, saturated model:       -286.0741

Information Criteria:

BIC, estimated model: 706.0751
BIC, saturated model: 1178.025

TestStatistics:

Likelihood ratio:   62.36662   p-val: 0.9999602
Pearson Chi^2:      92.6522   p-val: 0.9083178
Degress of freedom: 112
```

Although Agresti presents the results for three latent classes, the fit of this model using the `lca` function is somewhat unstable, as it does not appear to converge reliably. Actually, this is true for four classes as well. So, I use the two-class model for obtaining predicted counts and probabilities of positive diagnosis for each pathologist.

To obtain predicted probabilities of a positive diagnosis for each pathologist, for the two-class model, we can extract the `p` attribute from `res.lca`.

```
round(res.lca$p,3)

      1     2     3     4     5     6     7
1 1.000 0.983 0.761 0.541 0.979 0.423 1.000
2 0.117 0.354 0.000 0.000 0.223 0.000 0.117
```

To obtain fitted counts, one can use these predicted conditional probabilities. For example, the conditional probability of positive ratings by all seven pathologists given class 1 is the product of the first row of probabilities above. The analogous conditional probability for class 2 is the product of the second row. Thus, the unconditional probability of all positives is the sum of each of these products times the corresponding probability of the class: `res.lca$classprob`:

```
sum(counts)*(prod(res.lca$p[1,])*res.lca$classprob[1] +
prod(res.lca$p[2,])*res.lca$classprob[2]) # counts comes from above

[1] 9.9024
```

Function mmlcr

The package `mmlcr` (mixed-mode latent class regression) fits latent class regression models, including longitudinal models. The idea is that different latent classes could have different longitudinal trajectories or different regressions, involving the same form of regression, but with different coefficients. Also, multiple responses can be modeled. The different "responses" here are the seven pathologist ratings, and the "individuals" are the 118 slides. So, we build a new data frame out of `table.13.1`.

```
table.13.1.df<-data.frame(table.13.1[,1:7])
names(table.13.1.df)<-paste("p",1:7,sep="")
table.13.1.df<-sapply(table.13.1.df, as.factor) # make responses factors

# add an id variable
table.13.1.df<-data.frame(table.13.1.df[,1:7],id=1:nrow(table.13.1.df))
```

Now we fit a 2-class model using the function `mmlcr`. The `outer` argument gives a formula that describes class membership through covariates (i.e., what covariates predict latent class?). Here, we don't have covariates. The components argument is a list of the responses. Each response is a multinomial variable (binomial) evaluated once.

```
library(mmlcr)

# 2-class model
res.mmlcr<-mmlcr(outer = ~ 1 | id, components=list(
list(formula=p1~1, class= "multinomonce"),
list(formula=p2~1, class="multinomonce"),
list(formula=p3~1, class="multinomonce"),
list(formula=p4~1, class="multinomonce"),
list(formula=p5~1, class="multinomonce"),
list(formula=p6~1, class="multinomonce"),
list(formula=p7~1, class="multinomonce")), data=table.13.1.df, n.groups=2)

res.mmlcr

Coefficients:
  (Intercept)
1 0.000000000
2 0.005047976

Class Percentages:
   1    2
49.9 50.1

AIC: 666.8835
BIC: 737.6324
loglikelihood: -318.4418
```

Fitting more classes appears to be more stable with `mmlcr`. For example, fitting three classes gives

```
res3.mmlcr<-mmlcr(outer = ~ 1 | id, components=list(
list(formula=p1~1, class= "multinomonce"),
```

```
list(formula=p2~1, class="multinomonce"),
list(formula=p3~1, class="multinomonce"),
list(formula=p4~1, class="multinomonce"),
list(formula=p5~1, class="multinomonce"),
list(formula=p6~1, class="multinomonce"),
list(formula=p7~1, class="multinomonce")), data=table.13.1.df, n.groups=3)

res3.mmlcr

Coefficients:
  (Intercept)
1    0.000000
2    0.897036
3    0.726677

Class Percentages:
   1    2    3
18.1 44.4 37.5

AIC: 652.6177
BIC: 761.0994
loglikelihood: -303.3089
```

Predicted probabilities are somewhat more of a mystery out of `mmlcr`. You might think that the `fitted` component of the `mmlcr` object would give predicted probabilities of carcinoma from each pathologist for each of the classes, as in Agresti's Table 13.3. But, for example,

```
unique(sapply(res3.mmlcr$components,function(x) round(x$fitted-1,3))) # I use fitted – 1 because I used labels of 1,2 instead of 0,1 for absence/presence

      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]
 [1,] 0.058 0.142 0.000 0.000 0.056 0.000 0.000
 [2,] 0.119 0.256 0.000 0.008 0.149 0.000 0.085
 [3,] 0.391 0.765 0.000 0.043 0.564 0.000 0.463
 [4,] 0.466 0.905 0.000 0.052 0.679 0.000 0.568
 [5,] 0.517 1.000 0.000 0.059 0.756 0.000 0.638
 [6,] 0.514 0.994 0.000 0.058 0.751 0.000 0.633
 [7,] 0.632 0.995 0.204 0.184 0.814 0.114 0.724
 [8,] 1.000 0.981 0.859 0.587 1.000 0.477 1.000
 [9,] 0.940 0.983 0.753 0.522 0.970 0.418 0.955
```

The above gives 9 "classes" of probabilities, of which there appear to be 3 sets of similar probabilities, notably rows 1 and 2; rows 3, 4, 5, 6, and 7; and rows 8 and 9. These correspond roughly to the three sets of predicted probabilities in Agresti's Table 13.3.

Once the predicted carcinoma probabilities are obtained, expected counts can be gotten using the class percentages, as in the illustration for the function `lca`.

Function flexmix

The function `flexmix` in the R package of the same name fits finite mixtures of latent class regressions (generalized linear models) using an EM algorithm. To use it, we first `reshape` table.13.1 into the "long" format, and make the time variable the pathologist. An "id" variable indicates the subject (here, slides).

```
table.13.1.df<-data.frame(table.13.1[,1:7])
names(table.13.1.df)<-paste("p",1:7,sep="")
```

```
table.13.1.long<-reshape(table.13.1.df, varying=list(names(table.13.1.df)),
direction="long", v.names="response", timevar="pathologist")
```

Then, we make the variable `pathologist` a factor.

```
table.13.1.long$pathologist<-factor(table.13.1.long$pathologist)
```

Now, we use `stepFlexmix`, which operates just like the function `flexmix`, but is easier to use for multiple binary responses, like we have. (The "step" part applies when we want to do stepwise fits for several different numbers of components). First, we fit 2 components (`k` = 2).

```
library(flexmix)

fit1<-stepFlexmix(cbind(response,1-response)~pathologist|id,
data=table.13.1.long, k=2, model=FLXglm(family="binomial"), nrep=5)

summary(fit1)
```

```
       prior size post>0 ratio
Comp.1 0.499  413    462 0.894
Comp.2 0.501  413    455 0.908

'log Lik.' -317.2573 (df=15)
AIC: 664.5146   BIC: 735.2635
```

The `summary` function applied to the `flexmix` object gives the estimates of the component probabilities (`prior`), plus the number of observations assigned to each component (`size`). (The size must be divided by 7, to account for the 7 pathologists ratings per subject). The column "`post > 0`" tells you how many observations had posterior probabilities for the indicated component greater than a delta (again, must divide this column by 7 for this example), and "`ratio`" tells you the ratio of `size` to "`post > 0`". If components are well-separated, then many observations will have high posterior probabilities for one component, but low posterior probabilities for any other components. So, a high ratio will indicate well-separated components. We have fairly well-separated components for the pathologist example.

Calling `cluster(fit1)` will give the component assignments.

Now, we fit three components.

```
fit2<-stepFlexmix(cbind(response,1-response)~pathologist|id,
data=table.13.1.long, k=3, model=FLXglm(family="binomial"), nrep=5)

summary(fit2)
```

```
       prior size post>0 ratio
Comp.1 0.445  357    406 0.879
Comp.2 0.182  161    217 0.742
Comp.3 0.374  308    364 0.846

'log Lik.' -293.7051 (df=23)
AIC: 633.4101   BIC: 741.8918
```

Finally, the `plot` method for `flexmix` objects provides a rootogram.

2. Latent Class Models for Capture-Recapture

Latent class models can be used to estimate population size for a capture-recapture model. In Chapter 12, we used a generalized linear mixed model for this estimation. This model assumed a continuous mixture of capture probabilities. A two-class latent class model, for example, assumes two classes of capture probabilities. Each subject belongs in at most one class, and every subject in a class has the same capture probabilities.

For the snowshoe hares data in Table 12.6 in Agresti, we fit a two-class latent model using R. There are $T = 6$ occasions, and $2^T = 64$ cells, for each latent class. We proceed by building the design matrix. The full design matrix will have 128 rows, for the 64 cells for each class. There will be one column for each occasion, plus a column for a class variable, and a column for an intercept. We also add columns for the interaction between occasion and class.

```
i<-rep(1,128)
x<-as.integer(gl(2,64,128))-1
a<-as.integer(gl(2,32,128))-1
b<-as.integer(gl(2,16,128))-1
c<-as.integer(gl(2,8 ,128))-1
d<-as.integer(gl(2,4 ,128))-1
e<-as.integer(gl(2,2 ,128))-1
f<-as.integer(gl(2,1 ,128))-1

X<-cbind(i,x,a,b,c,d,e,f,x*cbind(a,b,c,d,e,f))

colnames(X)<-
c("Int","X","A","B","C","D","E","F","AX","BX","CX","DX","EX","FX")
```

The vector `counts` gives the frequencies for all rows except the all-zero row.

```
counts<-c(3,6,0,5,1,0,0,3,2,3,0,0,1,0,0,4,2,3,1,0,1,0,0,1,0,0,0,0,0,0,0,
4,1,1,1,2,0,2,0,4,0,3,0,1,0,2,0,2,0,1,0,1,0,1,0,1,1,1,0,0,0,1,2)
```

Now, we proceed in one of two ways. Of course, one way is to program the EM algorithm using the formula in section 13.1 of Agresti. Here, I try to take advantage of some existing R functions, namely the `emgllm` function from the `gllm` library. As explained in its help file, this function "fits log-linear models for incomplete contingency tables, including some latent class models, via an EM approach." Here, though, we are missing one cell's count. So, we place this function into the function `optim`, and try to estimate the missing count by minimizing the deviance.

```
library(gllm)

func<-function(y,s,X,counts){
    emgllm(c(y,counts),s,X,maxit=1)$deviance  # we only need 1 iteration
    }
```

The function `emgllm` takes several arguments, including a vector of cell indices, which we call "s". There are 64 cells, divided into two classes; so, `s` is given by

```
s<-c(1:64,1:64)
```

The first argument of `emgllm` is the vector of counts for the 64 cells. The first element of this vector is what we are estimating (the all-zero response). So, we leave this element as a variable called `y`. Now, we perform the optimization. In this case, the starting value can be just about anything. I use 2. The limits on the estimated count range from 0 to 100.

```
optim(par=c(2),func,control=list(REPORT=1,trace=3), lower=c(0), upper=c(100),
method="L-BFGS-B", s=s, X=X, counts=counts)

F = 58.3138
final  value 58.313787
converged
$par
[1] 7.066193

$value
[1] 58.31379

$counts
function gradient
       8        8

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

The estimated count is 7, which added to the the existing 68 gives a grand total of N=75 species, not quite the 85 obtained by Agresti. So, I try a second approach. With this approach, I write out the likelihood as per equations (13.2) and (13.1) in Agresti. Then, I calculate the deviance for optimization purposes. The parameters to optimize include the 12 conditional probabilities, $P(Y_t = y_t \mid Z = z)$, the class probabilities (just one, for two classes), and the missing count. The 14 paramters are denoted by `y`, in the function below. `X` is the design matrix (see below). `counts` has been defined above.

```
func<-function(y, X, counts) {

    n.s<-c(exp(y[14]),counts)

    z <- y[13]
    x<-1-y[1:12]

    res<-apply(X,1,function(w){ exp(sum(log(w*y[1:6] + (1-w)*x[1:6])))*z +
exp(sum(log(w*y[7:12] + (1-w)*x[7:12])))*(1-z)
    })

    xg0 <- n.s[n.s > 0]
    ll0 <- sum(xg0 * log(xg0/sum(n.s)))  # saturated likelihood

    ll<-n.s%*%log(res)

    -2 * (ll0 - ll)

}
```

First, I use the function `lca` (from R library e1071) to get starting values for the probabilities. Then, I minimize the deviance using `optim`.

.

```
library(e1071)
res<-lca(countpattern(temp[rep(1:63,counts),]),k=2) # starting values

X<-X[1:64,c("A","B","C","D","E","F")]     # see above for definition of X
```

```
result<-optim(par=c(as.numeric(res$p), res$classprob[1] ,log(20)),func,
control=list(fnscale=-1,trace=3,REPORT=1), lower=c(rep(0.001,13),0),
upper=c(rep(.999,13),100),X=X,counts=counts, method="L-BFGS-B")
$par
 [1] 0.1527328 0.2362195 0.1979381 0.1757010 0.2273388 0.1983629 0.4912584
 [8] 0.6181195 0.1576153 0.9990000 0.4043050 0.6391747 0.7564954 2.8362536

$value
[1] -41.23511

$counts
function gradient
      39       39

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

This result appears somewhat better, at least for estimating the missing count. It's estimate is exp(2.84), which is about 17. This number added to 68, gives an estimate for the total as 85, the same as Agresti obtained.

C. Nonparametric Random Effects Models

1. Logit Models with Unspecified Random Effects Distribution

The R package `npmlreg` performs nonparametric maximum likelihood estimation for generalized linear mixed models. The random effects distribution does not have to be specified a priori; however, the number of mass points must be pre-specified. The marginal likelihood is approximated using a finite mixture model, whose number of components equals the number of pre-specified mass points. The model can then be fitted using EM or other method for estimating finite mixture models. Thus, there is no numerical integration involved in the estimation. However, because there are no parametric distributional boundaries for the random effects distribution, the ML estimate can include positive probability at infinity, making it impossible to estimate a variance component.

Agresti fits a two-point nonparametric mixture distribution to the repeated binary measures in the attitudes toward abortion data (`table.10.13b`, above). We can fit this model using the function `allvc` which fits two-level hierarchical models. The form of the arguments is similar to that used by `glmmPQL` or `lme`, for example, where random effects are provided in a separate argument. Here we give a random intercept for each individual (`id`).

```
library(npmlreg)
res<-allvc(response ~ gender + question, random = ~1|id,
           family=binomial(link=logit),  data=table.10.13b, k=2)
summary(res)

Coefficients:
             Estimate Std. Error     t value
genderF   -0.02853461  0.1266663  -0.2252738
question2  0.30735602  0.1607048   1.9125504
question1  0.82846663  0.1591966   5.2040482
MASS1     -3.17213911  0.1513071 -20.9648995
MASS2      2.99145775  0.1497641  19.9744601
```

```
Mixture proportions:
    MASS1      MASS2
0.5289031  0.4710969
-2 log L:           4563.3     Convergence at iteration  9
```

The estimated two-point mixture distribution has mass points at about -3 and +3, with roughly equal probabilities. The function produces graphs of the trajectories that the estimates take as they converge to the final mass point estimates. The convergence appears to be relatively fast. Individual predicted probabilities range from very high to very low, with not many in between, due to the two separated random effects points. These are quite different from predictions from `glmmPQL`. A fit with three mass points puts the points at about -6, 0, and +6.

2. Nonparametric Mixing of Logistic Regressions

Adding a continuous covariate is not much more involved. We can use the `npmlreg` R package again, but this time we will use the `alldist` function instead of `allvc` because we will have only one level. We will also discuss the relative magnitude of the deviance reported by `alldist` (and `allvc`).

Table 13.4 in Agresti gives the number of protozoa exposed to a certain dose of poison, and the number that died. We read the data into R, first using counts.

```
table.13.4<-data.frame(dose=seq(4.7,5.4,.1),n=c(55,49,60,55,53,53,51,50),
dead=c(0,8,18,18,22,37,47,50))
```

In order to have alldist give the correct coefficient magnitudes, we expand the counts to create `table.13.4a`.

```
table.13.4a<-rbind(data.frame(dose=rep(table.13.4$dose,table.13.4$dead),
response=1), data.frame(dose=rep(table.13.4$dose, table.13.4$n-
table.13.4$dead), response=0))

library(npmlreg)
res<-alldist(response~log(dose), random=~1, family=binomial(link=logit), k=2,
data=table.13.4a)
summary(res)
```

```
Coefficients:
           Estimate Std. Error   t value
log(dose)  124.8910   15.42261  8.097919
MASS1     -205.8665   25.42324 -8.097573
MASS2     -196.3324   24.25993 -8.092867

Mixture proportions:
    MASS1      MASS2
0.6561909  0.3438091
-2 log L:           354.7     Convergence at iteration  50
```

One thing to notice is the deviance, which `alldist` reports as 354.7 (`res$deviance`), and Agresti reports as 3.4, coming from SAS. This is likely due to a negative constant factor not being included in the log likelihood from `alldist`. This is apparent when you fit an ordinary logistic, and note that the difference in deviances equals 21.3, which Agresti reports.

```
# ordinary logistic fit using alldist
res2<-alldist(response~log(dose), random=~1,family=binomial(link=logit), k=1,
data=table.13.4a)
```

```
res2$deviance-res$deviance
```

```
[1] 21.28307
```

```
res2$df.residual-res$df.residual
```

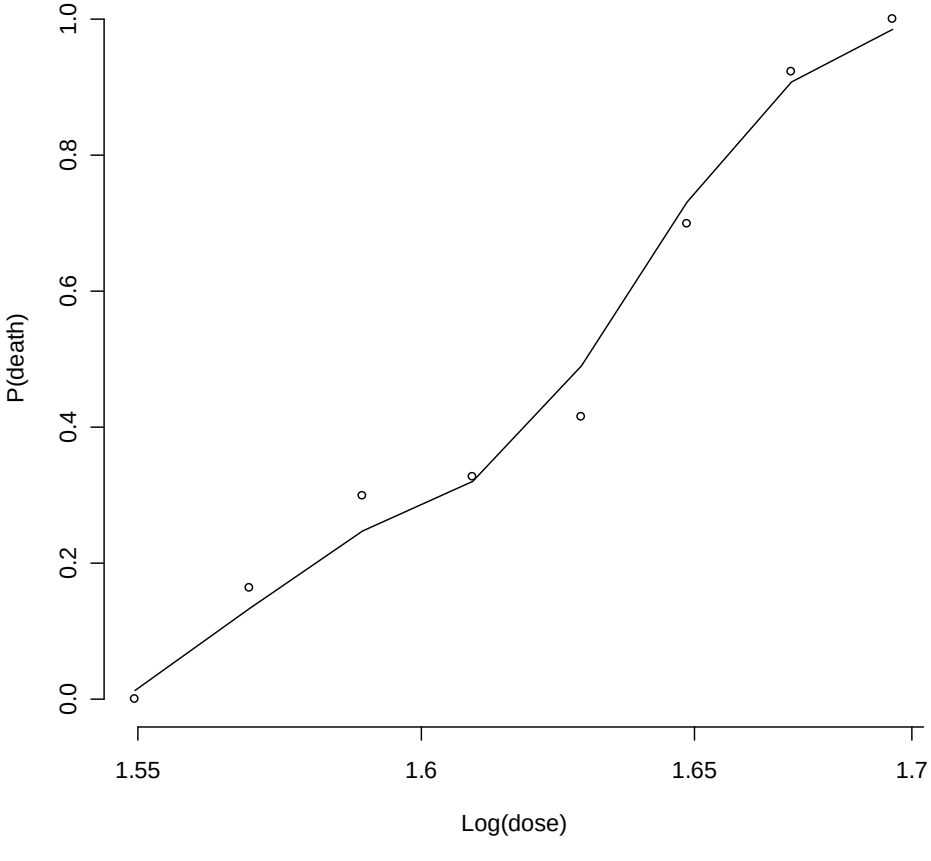
```
[1] 2
```

Predictions are obtained by fitting the binomial, instead of the binary responses.

```
res3<-alldist(cbind(dead,n-dead)~log(dose), random=~1, family=
binomial(link=logit), k=2, data=table.13.4)
ypred<-sort(predict.glmmNPML(res3,  type = "response"))
```

Now, we plot the observed proportions (circles) and a spline fit to the predictions, on the same graph.

```
plot(ldose<-log(table.13.4$dose),
table.13.4$dead/table.13.4$n,xlab="Log(dose)",ylab="P(death)",bty="L",axes=F)
axis(2)
axis(1, at=c(1.548,1.6,1.65,1.69,1.75),
labels=as.character(c(1.55,1.6,1.65,1.7,1.75)))
lines(smooth.spline(ldose,ypred,spar=.05))
```



3. Rasch Mixture Model

The Rasch model for a binary response for subject i and response t is

$$\text{logit}\left[P\left(Y_{it}=1\,|\,u_i\right)\right]=u_i+\beta_t,\ t=1,...,T$$

A Rasch mixture model assumes that the unobserved latent variable u_i has the discrete distribution

$$P\left(U=a_k\right)=\rho_k,\ k=1,...,q$$

The marginal probability of a sequence of responses $\left(y_1,...,y_T\right)$ is

$$\pi_{y_1,...,y_T}=\sum_{k=1}^{q}\rho_k\left[\prod_{t=1}^{T}\frac{\exp\left[y_t(a_k+\beta_t)\right]}{1+\exp(a_k+\beta_t)}\right]$$

This expression is then used in the appropriate multinomial likelihood in order to do estimation. Agresti states that once $q = (T + 1)/2$, the model gives the same fit as the quasi-symmetry model of Chapter 10.

Agresti uses the pathologist data, as well as some capture-recapture data to fit a Rasch mixture model. The pathologist data come from Table 13.1, which previously we put into a data frame called `table.13.1`. The "long" version of `table.13.1` we called `table.13.1.long`. This version has a column for "id", the slide.

We can use `allvc` from the `npmlreg` R package. It is very straightforward to fit a Rasch mixture model with 3 mass points (argument `k`). I use sum-to-zero contrasts for the beta coefficients in order to match Agresti's output.

```
options(contrasts=c("contr.sum", "contr.poly")) # sum-to-zero contrasts

library(npmlreg)
summary(fit.rm<-allvc(response~pathologist, random=~1|id,
family=binomial(link=logit), k=3, data=table.13.1.long))

Coefficients:
             Estimate Std. Error    t value
pathologist1  1.476153  0.4312912   3.422637
pathologist2  3.513361  0.5064987   6.936565
pathologist3 -1.868961  0.4434189  -4.214887
pathologist4 -3.154055  0.4621194  -6.825196
pathologist5  2.255030  0.4512639   4.997143
pathologist6 -3.697682  0.4659513  -7.935769
MASS1        -5.244664  0.4780734 -10.970416
MASS2        -1.015508  0.3261387  -3.113731
MASS3         3.632482  0.4124120   8.807897

Mixture proportions:
    MASS1      MASS2      MASS3
0.3710929  0.1944598  0.4344473
-2 log L:           599.6     Convergence at iteration  18
```

From the output, pathologist B (pathologist2) tends to make a carcinoma diagnosis most often. The numbers in Table 13.3 in Agresti give the estimated probabilities of a carcinoma diagnosis, conditional on each of the three latent classes. They are obtained by plugging the coefficient estimates above into the formula on p. 550 in Agresti.

4. Log-linear Models for Capture-Recapture

The Cormack log-linear model for capture-recapture is a marginal model with capture probabilities averaged over subjects. With T capture occasions, there are $2^T - 1$ observed cells. The idea is to estimate the unobserved cell count from the observed cell counts, using the estimated log-linear parameters.

We fit several log-linear models to the snowshoe hare data in Table 12.6 in Agresti. Two R packages that fit these models effortlessly are the `repeated` library and the `Rcapture` library, the latter being naturally more comprehensive. `Rcapture` has an associated reference article that explains many more features than what I will use (see the package's help file).

I will set up the data to work first with the `capture` function from the `repeated` library. Below are the counts.

```
counts<-c(0,3,6,0,5,1,0,0,3,2,3,0,0,1,0,0,4,2,3,1,0,1,0,0,1,0,0,0,0,0,0,
0,4,1,1,1, 2,0,2,0,4,0,3,0,1,0,2,0,2,0,1,0,1,0,1,0,1,1,1,0,0,0,1,2)
```

Now, we call up repeated, and then "`eval`" an expression called "`setup`", which forms a matrix with each row representing a capture history across the n = 6 occasions (1 = capture; 0 = no capture). This matrix is constructed by "`cbind`"ing n = 6 column vectors: `p1`…`p6`.

```
library(repeated)

n<-6  # number of occasions
eval(setup)
cbind(p1,p2,p3,p4,p5,p6)       # shows the matrix of capture histories
```

Because of the order of the rows in the above matrix, I reorder the `counts` vector to correspond to the correct capture histories. The next four lines do this, and call the new counts vector, `new.counts`.

```
reindex<-c(1, 5, 3, 7, 2, 6, 4, 8)
index<-outer(seq(0,56,by=8),reindex,"+")
index<-index[reindex,]
new.counts<-counts[as.numeric(index)]
```

First, we fit a model of mutual independence of occasions. As you can see, the "workhorse" is just the `glm` function with poisson family, as it is for log-linear models fit in previous chapters. Notice the case `weights` argument is `pw`. `pw` is a vector generated by the `eval` of `setup`. It gives a weight of 1 to observable histories, and a weight of 0 to the unobservable history. However, `setup` puts the 0 last, and according to our `new.counts` vector, we need it first. Hence, the call to `rev`.

```
z0 <- glm(new.counts~(p1+p2+p3+p4+p5+p6), family=poisson, weights=rev(pw))
```

Next, the call to `capture` prints out the estimated abundance after each occasion. It is not really useful for a closed population with independent occasions. You may consult the references in the help file for explanation of the remaining columns.

```
capture(z0,n)
```

```
     i    N(i) Phi(i-1)      P(i)         B(i-1)
[1,] 1 75.0662        1 0.5737096  7.506620e+01
[2,] 2 75.0662        1 0.6936038 -1.421085e-14
[3,] 3 75.0662        1 0.6536391  0.000000e+00
```

```
[4,] 4 75.0662        1 0.7335685 -1.421085e-14
[5,] 5 75.0662        1 0.6269959  0.000000e+00
[6,] 6 75.0662        1 0.7868548  0.000000e+00
```

Finally, the estimated abundance from the log-linear model is found by adding the estimated count in the first cell to the number of observed captures (68).

```
z0$fit[1]+68
```

```
75.0662
```

A model with all two-factor associations gives

```
z1 <- glm(new.counts~(p1+p2+p3+p4+p5+p6)^2, family=poisson, weights=rev(pw))
capture(z1,n)
```

```
     i     N(i)  Phi(i-1)      P(i)        B(i-1)
[1,] 1 81.77645 1.0000000 0.8899998  8.177645e+01
[2,] 2 76.92659 0.9406936 0.8877724 -1.421085e-14
[3,] 3 75.48739 0.9627109 0.8954631  1.429333e+00
[4,] 4 78.28810 0.9816676 0.9102879  4.184571e+00
[5,] 5 82.96017 1.0059834 0.8606738  4.203642e+00
[6,] 6 97.85445 1.0000000 0.9072763  1.489428e+01
```

In the above matrix, the `N(i)` column gives the estimate of the total population at the ith period. `Phi(i-1)` gives the probability that an individual survives from the (`i-1`)th to the `i`th period. `P(i)` gives the probability of capture. `B(i-1)` gives the number of individuals added to the population between the (`i-1`)th and `i`th periods.

The following, of course, gives the estimate of total abundance.

```
z1$fit[1]+68
```

```
104.7810
```

Using `Rcapture`, we can fit a number of models that allow for different types of association among occasions. A special case of a quasi-symmetry model where each pair of occasions has the same association (exchangeable association) can be fit using the `closedp` function from `Rcapture`. This function takes as its first argument, the matrix of capture histories, with the last column either including the frequencies of each capture history (`dfreq=TRUE`) or not (`dfreq=FALSE`). (Note that there are functions in Rcapture that will generate this matrix for you.)

```
library(Rcapture)
```

```
fit.cap<-closedp(cbind(p1,p2,p3,p4,p5,p6,new.counts),dfreq=TRUE)
```

```
Number of captured units: 68

Abundance estimations and model fits:
              abundance  stderr  deviance  df      AIC
M0                 75.4     3.5    68.516  61  154.707
Mt                 75.1     3.4    58.314  56  154.505
Mh Chao            79.8     6.4    58.023  58  150.214
Mh Poisson2        81.5     5.7    59.107  60  147.298
Mh Darroch         90.4    11.6    61.600  60  149.791
```

```
Mth Chao           79.6     6.3    47.115  52  151.305
Mth Poisson2       81.1     5.6    48.137  55  146.327
Mth Darroch        90.5    11.7    50.706  55  148.896
Mb                 72.2     3.1    67.133  60  155.323
Mbh                78.1    13.1    63.988  59  154.179

Note: 1 eta parameter has been set to zero in the Mh Chao model
```

The function will output abundance estimates for a variety of models. The documentation, in addition to the JASA article, explains them. Here, we will focus on the `Mth Darroch` model. This is the exchangeable association model. The abundance estimate is 90.5 hares. To get a profile likelihood confidence interval (along with a plot), we can use `profileCI` function with the Mth Darroch method.

```
profileCI(cbind(p1,p2,p3,p4,p5,p6,new.counts), dfreq=TRUE, m="Mth",
h="Darroch")

Number of captured units: 68

95% Profile likelihood confidence interval:
             abundance     InfCL     SupCL
Mth Darroch         88  73.64514  121.1492
```

The abundance estimate from the `profileCI` function differs from the one given by `closedp`.

We could, in fact, estimate abundance for the exchangeable model using only `glm`. To do this we must add another term to the formula that represents the association. The Darroch, et al (1993) article referenced in Agresti shows that this term for each capture history is the sum of the capture indicators "choose" 2 (pairs); which is zero when the sum is less than 2. Thus,

```
assoc<-choose(n=rowSums(cbind(p1,p2,p3,p4,p5,p6)),k=2)
(z1 <- glm(new.counts~p1+p2+p3+p4+p5+p6+assoc, family=poisson, weights=
rev(pw)))

Coefficients:
(Intercept)           p1           p2           p3           p4           p5
     3.1119      -1.4126      -1.9646      -1.7696      -2.1758      -1.6464
         p6        assoc
    -2.4932       0.2225

Degrees of Freedom: 62 Total (i.e. Null);  55 Residual
Null Deviance:      112.8
Residual Deviance: 50.71        AIC: 148.9

z1$fit[1] + 68

90.4641
```

This is the same fit as

```
fit.cap$glmMthD

Coefficients:
 (Intercept)  mXMthD.beta1  mXMthD.beta2  mXMthD.beta3  mXMthD.beta4
       3.112        -1.413        -1.965        -1.770        -2.176
mXMthD.beta5  mXMthD.beta6    mXMthD.tau
      -1.646        -2.493         0.445

Degrees of Freedom: 62 Total (i.e. Null);  55 Residual
```

```
Null Deviance:      112.8
Residual Deviance: 50.71        AIC: 148.9
```

but where `mXMthD.tau` is two times `assoc` parameter estimate.

5. Nonparametric Mixtures and Quasi-symmetry

Agresti shows that a distribution-free approach for the random effects with the Rasch mixture model implies the quasi-symmetry loglinear model marginally. The loglinear model will have terms for main effects for all T responses, plus a term for unique values of the sum of the 0/1 responses (the quasi-symmetry term). We illustrate using the data on opinions about legalized abortion analyzed in Section 13.2.1.

We had these data in a data frame called `table.10.13`.

`table.10.13`

```
   gender question1 question2 question3 count  symm
1       M         Y         Y         Y   342 Y,Y,Y
2       F         Y         Y         Y   440 Y,Y,Y
3       M         N         Y         Y     6 N,Y,Y
4       F         N         Y         Y    14 N,Y,Y
5       M         Y         N         Y    11 N,Y,Y
6       F         Y         N         Y    14 N,Y,Y
7       M         N         N         Y    19 N,N,Y
8       F         N         N         Y    22 N,N,Y
9       M         Y         Y         N    26 N,Y,Y
10      F         Y         Y         N    25 N,Y,Y
11      M         N         Y         N    21 N,N,Y
12      F         N         Y         N    18 N,N,Y
13      M         Y         N         N    32 N,N,Y
14      F         Y         N         N    47 N,N,Y
15      M         N         N         N   356 N,N,N
16      F         N         N         N   457 N,N,N
```

We will convert columns 2 through 4 into 0/1 variables.

```
table.10.13c<-table.10.13[,2:4]
table.10.13c<-as.data.frame(lapply(table.10.13c,function(x) as.numeric(x)-1))
```

Then, we add a term for quasi-symmetry, called `symm`. It will contain the row sums for each cell combination.

```
symm<-as.factor(rowSums(table.10.13c))
table.10.13c<-data.frame(table.10.13c, count=table.10.13$count, gender=
table.10.13$gender, symm=symm)
```

Then, we call `glm` to fit the loglinear model. I don't use an intercept to match equation (13.7) in Agresti.

```
summary(glm(count~-1+question1+question2+question3+gender+symm,
family=poisson, data=table.10.13c))
```

```
Coefficients: (1 not defined because of singularities)
          Estimate Std. Error z value Pr(>|z|)
question1  0.43689    0.09137   4.781 1.74e-06 ***
question2 -0.08420    0.09202  -0.915     0.36
question3 -0.39156    0.09512  -4.117 3.84e-05 ***
```

```
genderM     5.87852    0.04381 134.176  < 2e-16 ***
genderF     6.12188    0.04067 150.537  < 2e-16 ***
symm1      -2.77704    0.08653 -32.094  < 2e-16 ***
symm2      -3.26477    0.10747 -30.379  < 2e-16 ***
symm3            NA         NA      NA       NA
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 17076.229  on 16  degrees of freedom
Residual deviance:    10.156  on  9  degrees of freedom
AIC: 112.84
```

Note also that for the `symm` sum, only the values 1 and 2 provide additional information. The value 0 provides no more information than does knowing the fact that all three responses are 0. Similarly, the value 3 provides no more information than does knowing that all three responses are 1. Thus, we have no estimates for terms corresponding to these values.

D. Beta-Binomial Models

For a binomial random variable, Y, with index n and probability π, if π is distributed beta with mean $\mu = \alpha/(\alpha+\beta)$ and variance $\mu(1-\mu)\theta/(1+\theta)$, where $\theta = 1/(\alpha+\beta)$, then marginally, Y has a beta-binomial distribution with mean $n\mu$ and variance $n\mu(1-\mu)[1+(n-1)\theta/(1+\theta)]$. One can incorporate covariates or predictors within a beta-binomial model via a link function of μ. For example, for covariates $\mathbf{x}$, one could model the relationship $\text{logit}(\mu) = \alpha_0 + \boldsymbol{\beta}^T\mathbf{x}$.

The beta-binomial can be used to model overdispersed counts. (One can see that the variance of a beta-binomial count is at least as large as the variance of a binomial count.) Another way to model overdispersed counts is to use quasi-likelihood with variance function similar to the beta-binomial variance. See equation (13.10) in Agresti, which also defines the "overdispersion parameter".

Estimation of θ and $\boldsymbol{\beta}$ (and α_0) can be done using maximum likelihood, alternating between fixing θ and maximizing over $\boldsymbol{\beta}$, and vice versa. Alternatively, one can use an approach due to Williams that alternates between solving the quasi-likelihood equation for $\boldsymbol{\beta}$ for a given value of the "overdispersion parameter", and solving for the overdispersion parameter using an equation that sets the Pearson chi-squared statistic equal to the residual degrees of freedom for the model. Finally, perhaps the simplest approach was introduced in Chapter 4, where the variance function is defined as $v(\mu) = \phi n\mu(1-\mu)$, and the overdispersion parameter, ϕ, is estimated using $\hat{\phi} = X^2/df$.

In Chapter 4, we fit an overdispersed binomial model to the teratology data in Table 4.5 in Agresti. We did that using quasi-likelihood with variance function $v(\mu) = \phi n\mu(1-\mu)$. Here we will fit beta-binomial models, as well as quasi-likelihood models with variance function similar to the beta-binomial variance. First, we read in the data from a text file, and set it up as a data frame.

```
table.4.5<-read.table("teratology.txt", col.names=c("","group", "litter.size",
"num.dead"))[,-1]
table.4.5$group<-as.factor(table.4.5$group)

group<-table.4.5$group
```

Now, we can fit a beta-binomial model, with estimation of parameters via maximum likelihood (or some other variant that uses the beta-binomial likelihood directly). We will use three R packages: VGAM, gnlm, and aod (analysis of overdispersion). The last one has not yet been used in the manual.

We have used the vglm function from VGAM before. Here, we just change the family argument to betabinomial(zero=2, irho=.2). The zero argument indicates which of the two parameters (if any) should have zero covariates. The parameters in reference are the mean (mu) and the correlation parameter (rho). One can also model the logit of the correlation parameter to depend on covariates given after the ~ in the formula argument (in that case use zero=NULL). The argument irho specifies the initial value for rho.

The formula in the first argument gives the linear model for the logit, here of just mu, and not also of rho. Incidentally, there are also arguments within betabinomial to change the link function from logit to another link (using lmu and lrho).

```
library(VGAM)
fit.bb<-vglm(cbind(num.dead,litter.size-num.dead) ~ group,
betabinomial(zero=2, irho=.2), data=table.4.5, trace=TRUE)

summary(fit.bb)

Coefficients:
                Value Std. Error t value
(Intercept):1  1.3459    0.24412  5.5132
(Intercept):2 -1.1459    0.32408 -3.5360
group2        -3.1143    0.51836 -6.0079
group3        -3.8678    0.86346 -4.4794
group4        -3.9225    0.68357 -5.7383

Number of linear predictors:  2

Names of linear predictors: logit(mu), logit(rho)

Dispersion Parameter for betabinomial family:   1

Log-likelihood: -219.3454 on 111 degrees of freedom

Number of Iterations: 7
```

So, we have two linear predictors. The (Intercept):2 value is the logit of rho. Thus, rho is estimated as the inverse logit of -1.1459.

```
logit(-1.146, inverse=T)

[1] 0.2412205
```

The R package gnlm has a general function called gnlr which fits many likelihoods, including beta-binomial. We've seen this function in other chapters. For the argument mu, we will use a function. The function must have an argument, p, for "parameter" (see the help file for using a function of the elements of vector p). Our function for mu also has an argument, linear, which is where the linear predictor goes. Here, we use the (anti) logit function from VGAM, hence the require call. We could have used any (inverse) link function, but we use logit here. The arguments, pmu and pshape give initial values. The initial values for the coefficients in the linear predictor will come from the previous fit we did.

```
library(gnlm)
```

```
require(VGAM)   # for logit function

mu <- function(p, linear) logit(linear,inverse=T)

gnlr(cbind(table.4.5$num.dead,table.4.5$litter.size-table.4.5$num.dead),
distribution="beta binomial",  linear=~group, mu=mu, pmu=
coefficients(fit.bb)[c(1,3:5)], pshape=.25)

beta binomial distribution

Response:       cbind(table.4.5$num.dead,       table.4.5$litter.size       -
table.4.5$num.dead)

Log likelihood function:
{
    m <- mu1(p)
    s <- exp(sh1(p))
    t <- s * m
    u <- s * (1 - m)
    -sum(wt * (lbeta(y[, 1] + t, y[, 2] + u) - lbeta(t, u)))
}

Location function:
logit(linear, inverse = T)
Linear part:
~group

Log shape function:
p[1] * rep(1, n)

-Log likelihood    93.45675
Degrees of freedom 53
AIC                98.45675
Iterations         11

Location parameters:
            estimate      se
(Intercept)     1.346  0.2482
group2         -3.114  0.5018
group3         -3.868  0.8081
group4         -3.923  0.6675

Shape parameters:
      estimate      se
p[1]     1.146  0.3299
```

From the output, the shape parameter estimate is claimed to be the log of the correlation parameter. However, from our last fit, we see that it is really the negative of the logit of the correlation parameter. That is, -logit(0.24) = `p[1]`.

Finally the R package `aod` has a function called `betabin`, which can fit a beta-binomial likelihood. The `random` argument is actually a formula for the dispersion parameter (Agresti's rho). For example, if we had specified `~group`, then we would get a different dispersion parameter per group. Here, we make the formula constant.

```
library(aod)
betabin(cbind(num.dead,litter.size-num.dead)~group,random=~1,data=table.4.5)
```

```
Beta-binomial model
-------------------

Fixed-effect coefficients:
             Estimate Std. Error    z value Pr(> |z|)
(Intercept)  1.346e+00  2.481e-01  5.425e+00 5.799e-08
group2      -3.115e+00  5.020e-01 -6.205e+00 5.485e-10
group3      -3.869e+00  8.088e-01 -4.784e+00 1.722e-06
group4      -3.924e+00  6.682e-01 -5.872e+00 4.293e-09

Overdispersion coefficients:
                 Estimate Std. Error   z value   Pr(> z)
phi.(Intercept) 2.412e-01  6.036e-02 3.996e+00 3.222e-05

Log-likelihood statistics
   Log-lik      nbpar    df res.   Deviance        AIC       AICc
-9.346e+01          5         53  1.154e+02  1.969e+02  1.981e+02
```

To fit the quasi-likelihoods that Agresti fits, we can also use package `aod`. The function `quasibin` in `aod` can fit the QL(1) model, with beta-binomial variance.

```
require(aod)
quasibin(cbind(num.dead,litter.size-num.dead)~group,data=table.4.5) # QL(1)

Quasi-likelihood generalized linear model
-----------------------------------------

Fixed-effect coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   1.2124     0.2233  5.4294   < 1e-4
group2       -3.3696     0.5626 -5.9893   < 1e-4
group3       -4.5853     1.3028 -3.5197    4e-04
group4       -4.2502     0.8484 -5.0097   < 1e-4

Overdispersion parameter:
   phi
0.1923

Pearson's chi-squared goodness-of-fit statistic = 54.0007
```

The function `bnlr` in R package `gnlm` fits binomial non-linear regression models with a variety of link functions. With the logit link, we get the Binomial ML results in Table 13.5 in Agresti.

```
library(gnlm)
(fit.ml<-bnlr(cbind(table.4.5$num.dead,table.4.5$litter.size-
table.4.5$num.dead), link="logit", mu=~group,pmu=c(1,1,1,1)))

binomial distribution

Response: cbind(table.4.5$num.dead, table.4.5$litter.size -
table.4.5$num.dead)

Log likelihood function:
{
    m <- plogis(mu1(p))
    -sum(wt * (y[, 1] * log(m) + y[, 2] * log(1 - m)))
}
```

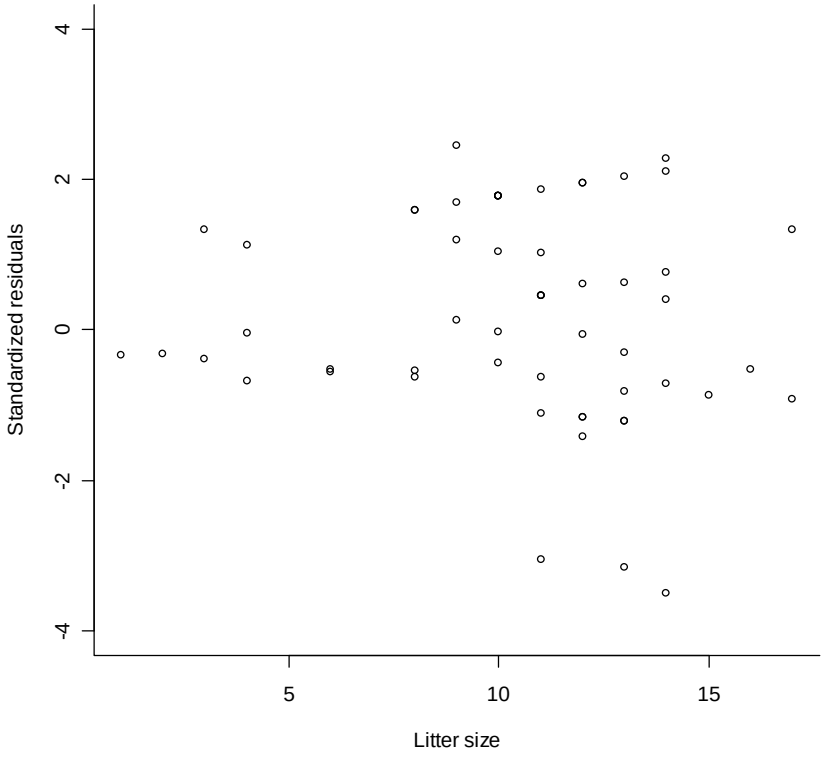
```
Location function:
~group

-Log likelihood    122.4613
Degrees of freedom 54
AIC                126.4613
Iterations         20

Location parameters:
             estimate      se
(Intercept)     1.144  0.1292
group2         -3.323  0.3308
group3         -4.476  0.7310
group4         -4.130  0.4761

Correlations:
        1        2        3        4
1  1.0000 -0.39060 -0.17676 -0.27138
2 -0.3906  1.00000  0.06904  0.10600
3 -0.1768  0.06904  1.00000  0.04797
4 -0.2714  0.10600  0.04797  1.00000
```

For any of these fits, the probability of death is much lower for any of the treatments (group2, group3, and group4) than for the placebo (Intercept only).

Agresti mentions a diagnostic used to determine whether the beta-binomial variance function is more appropriate with a quasi-likelihood model than is the simpler variance function. If a plot of the standardized Pearson residuals for the ordinary binomial model against the indices n_i shows an increasing trend in spread as n_i increases, then a beta-binomial variance function might be more suitable.

Next, I show this plot for the teratology data. I compute the standardized residuals from the `fit.ml` object, as it fit an ordinary binomial. For `bnlr`, the call function, there is no simple way to extract standardized Pearson residuals (as there is with `glm` objects). So, I must do a little "hand" calculation.

```
logit.inverse<-function(x) exp(x)/(1+exp(x))
n<-table.4.5$litter.size
Pi<-logit.inverse(fitted(fit.ml)/n)
r<-table.4.5$num.dead-n*Pi
std.pear.res<-r/sqrt(n*Pi*(1-Pi))

plot(n, std.pear.res, ylim=c(-4,4), ylab="Standardized residuals",
xlab="Litter size", bty="L")
```



From the plot, it is pretty apparent that the spread of the residuals increases with litter size. Thus, the beta-binomial variance function might be most appropriate.

E. Negative Binomial Regression

A regression model for count data (e.g., Poisson regression) might show overdispersion (variance of response exceeds the mean) if some relevant explanatory variables aren't in the model. The Negative Binomial is a gamma mixture of Poissons so that given a rate, say λ, the count Y has a Poisson distribution with mean λ, and λ has a gamma distribution with shape and scale parameters, k and μ. The marginal distribution for Y, then, is negative binomial with mean μ and dispersion parameter k^{-1}. As dispersion goes to zero, the negative binomial becomes a Poisson. We saw the negative binomial probability distribution function in Chapter 4.

Negative binomial regression models have (a function of) μ depend on predictor variables. For a fixed k, the negative binomial is in the exponential family. Thus, one can use IRLS for estimation of regression parameters, given k, then alternate to estimate k, iterating between them until convergence.

Table 13.6 in Agresti gives responses from 1308 persons to the question: Within the past 12 months, how many people have you known personally who were victims of homicide? The table shows responses by race (1149 whites and 159 blacks). Agresti notes that the sample response means for each race (p. 561) are roughly double the sample variances.

A Poisson generalized linear model can be fit using `glm` in both R and S-PLUS. Agresti gives the results from that fit, and shows underfitting of counts at response = 0. There are actually several R packages

with functions for fitting negative binomial regression models, and special cases. Probably the most typically used is `glm.nb` in `MASS`, which I use below.

The data from Table 13.6 are on Agresti's website. However, I have also saved the data in a file called `homicide.txt` on my website (where you got this manual). So, we read it in, and use only columns 1, 2, and 4.

```
homicide.fr<-read.table("homicide.txt", header=TRUE)
homicide.fr<-homicide.fr[,c(1,2,4)]
```

Then, I do a little manipulation to put it in a form useable by `glm.nb`, so that response is a column, and the frequency of that response by race is recorded.

```
homicide.fr<-data.frame(freq=c(homicide.fr[,1],homicide.fr[,2]),
response=rep(homicide.fr[,3],2), race=rep(c("white","black"),each=7))
homicide.fr$race<-factor(homicide.fr$race, levels=c("white","black"))

homicide.fr
```

```
   freq response  race
1  1070        0 white
2    60        1 white
3    14        2 white
4     4        3 white
5     0        4 white
6     0        5 white
7     1        6 white
8   119        0 black
9    16        1 black
10   12        2 black
11    7        3 black
12    3        4 black
13    2        5 black
14    0        6 black
```

Then, we fit the negative binomial model, with a log link for the mean, using the frequencies as weights.

```
require(MASS)
fit.nb<-glm.nb(response~race, data=homicide.fr, weights=freq, link=log)
summary(fit.nb)


Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -2.3832     0.1172 -20.335  < 2e-16 ***
raceblack     1.7331     0.2385   7.268 3.66e-13 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for Negative Binomial(0.2023) family taken to be 1)

    Null deviance: 471.57  on 10  degrees of freedom
Residual deviance: 412.60  on  9  degrees of freedom
AIC: 1001.8

Number of Fisher Scoring iterations: 1

Correlation of Coefficients:
```

```
          (Intercept)
raceblack -0.49


              Theta:  0.2023
          Std. Err.:  0.0409
```

As discussed in Agresti, the fitted coefficient of 1.733 is the log of the ratio of the sample response means for black and white (i.e., 1.733 = log(0.522/0.092). The estimated value of theta, 0.2023, is the same as the estimate of *k*, in Agresti's text. This can be used to estimate the variance of the response, per race, as shown in Agresti.

The predicted mean response per race is the sample mean:

```
unique(predict(fit.nb, type="response"))
[1] 0.09225413 0.52201258
```

We can also get predicted counts per response for each race, using the density function (probability function) for the negative binomial. For whites, they are:

```
round(dnbinom(0:6, size=fit.nb$theta, mu=.092)*1149,1)

[1] 1065.1   67.4   12.7    2.9    0.7    0.2    0.1
```

And, for blacks they are:

```
round(dnbinom(0:6, size=fit.nb$theta, mu=.522)*159,1)


[1] 122.8  17.9   7.8   4.1   2.4   1.4   0.9
```

Compare these to Table 13.6 in Agresti.

F. Poisson Regression with Random Effects

A common generalized linear mixed model for count responses is the Poisson regression model with random intercept. As before, the random intercept is often given a normal distribution. Thus, the mean for observation t in cluster i is

$$\log\left[E(Y_{it} \mid u_i)\right] = \mathbf{x}_{it}^T\boldsymbol{\beta} + u_i$$

where $u_i \sim N(0, \sigma^2)$

Conditional on u_i, y_{it} has a Poisson distribution. Marginally, the distribution has variance greater than the mean whenever σ > zero. The ordinary Poisson model results when the normal variance is zero, that is, no random effects.

To compare with the previous section, we fit a Poisson generalized linear mixed model to the Homicide data found on Agresti's website. We had previously put that text file into a data frame called `homicide.fr`. To use some of the functions for fitting a generalized linear mixed model, we "expand" the data frame so that each row corresponds to one subject, called `ID`.

```
homicideA.fr<-data.frame(homicide.fr[rep(1:14,homicide.fr$freq),2:3],
ID=factor(1:sum(homicide.fr$freq)))
```

There are several R/S functions for fitting Poisson generalized linear mixed models. One function whose default results match those of SAS NLMIXED (which are given in Agresti) is the function `glmmML`, in the R package of the same name. Here, I use it on the `homicideA.fr` data frame, with the Gaussian quadrature method.

```
require(glmmML)
res.glmmML<-glmmML(response ~ race, cluster=ID,family=poisson, method="ghq",
data= homicideA.fr)
```

```
              coef se(coef)       z Pr(>|z|)
(Intercept) -3.689   0.2452 -15.046 0.00e+00
raceblack    1.890   0.2453   7.705 1.31e-14

Scale parameter in mixing distribution:  1.632 gaussian
Std. Error:                              0.1588

Residual deviance: 728.1 on 1305 degrees of freedom     AIC: 734.1
```

The fitted marginal means and variances can be obtained by accessing the coefficients (e.g., `res.glmmML$coefficients` and `res.glmmML$sigma`). The fitted counts from the model, however, are not obtained from a Poisson distribution because the marginal distribution is not Poisson.

Additional R functions that can be used to fit Poisson generalized linear mixed models are `glmer` from the `lme4` package and `glmmPQL` from the `MASS` package. Here are some example calls.

```
require(lme4)
glmer(response ~ race + (1 | ID) , family = poisson(log), data = homicideA.fr,
nAGQ=2, REML=F,verbose=T)
```

```
require(MASS)
glmmPQL(response~race, random=~1|ID, data=homicideA.fr, family=poisson(log),
verbose=T, start=c(-3.7,1.9))
```