R COMPANION FOR DESIGN AND ANALYSIS OF EXPERIMENTS

Version 2021-03-10 for the 10th edition.

Edgar Hassler

Contents

Preface

The goal of this document is to provide a companion guide for the use of R with the textbook *Design and Analysis of Experiments*, 10th ed. by Douglas C. Montgomery. The textbook includes many examples using Minitab, Design Expert, and JMP software, and I have tried to reproduce these analyses to the best of my abilities. However, some differences were unavoidable, and some capabilities do not yet exist in R. By the time you read this new packages may have been developed and new capabilities added to existing packages. A good summary of the state of experimental design in R is maintained at https://CRAN.R-project.org/view=ExperimentalDesign.

I've created an R package, `MontgomeryDAE`, that contains the data sets used in the textbook as well as this document. It is hosted on GitHub at https://github.com/ehassler/MontgomeryDAE. If you find errors or have suggestions you can post them as issues in GitHub.

Edgar Hassler

Tempe, Arizona
November, 2019

Chapter 1

A Taste of R

R is an interpreted language for analyzing and visualizing data. It's very different from products like Minitab and JMP even though the users' goals are often the same. Perhaps the most important factor for the success of R is that it's free and open source. Because of its accessibility, many people have chosen to adopt it for research, business purposes, and teaching. Some of them have created packages that would let R do the analysis they needed to do, or as a way to enable people to do a new kind of analysis altogether. R grew and grew. Today it is an indispensable tool for many statisticians and data scientists.

There's no way to cover most or even much of R and the important packages. Instead, I've tried to provide just a taste of R, enough to get people through the examples contained in the subsequent sections. Each following chapter corresponds to the same numbered chapter in *Design and Analysis of Experiments*. This document is very much a companion — on its own it will likely make little to no sense, but used with the textbook it should allow R users to follow along with the examples.

This chapter is divided into two main parts. First, I will describe how to get and install R and associated tools. Second, I will cover how key features of the R language work.

Getting Started

To get started we will first download the R interpreter so that we can run R code. Then we will download the RStudio editor to make editing nice and easy. Finally we will install the `MontgomeryDAE` package from GitHub to bring in all of the data sets from the textbook.

Installing the R Interpreter

At present you can find R through the website https://www.r-project.org. There you can download a version of R best suited for your operating system, and learn what code names for each release. (The present release 3.6.1 is *Action of the Toes*.)

For macOS users, instead of installing R from the `pkg`, I'd recommend installing homebrew and installing R using homebrew. At present you can do this by running the following from Terminal:

```
/usr/bin/ruby -e "$(curl -fsSL
↪  https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Next, to install R using homebrew simply run the following in the terminal:

```
brew install r
```

The homebrew installation method makes things such as installing R source packages easier down the road. Windows users can install `Rtools` to provide a toolchain for compiling source packages as well.

When reading about R you'll see CRAN a lot. CRAN standards for Comprehensive R Archive Network and it's where you'll actually be downloading R and most packages. Packages are collections of code (in R, C, Fortran, or other languages), data sets, and documentation that we load using the `library` function. There is a *huge* variety of R packages that cover almost anything you could want (note the "almost" qualifier).

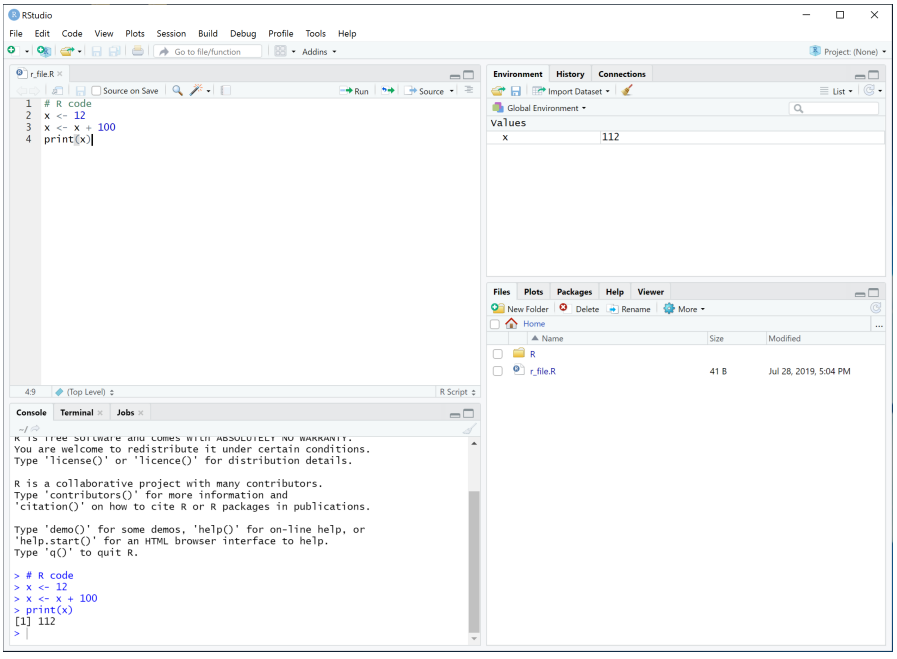
Microsoft began seriously investing in R back in 2015 when it acquired Revolution Analytics. Since then they've developed Microsoft R Open (MRO) which is their own runtime for R that has improved performance (see https://mran.microsoft.com/open) but often lags behind the current version of CRAN R. Microsoft has integrated R into other products as well — you can run R code directly inside SQL Server queries. I tend to use CRAN R over MRO since that's the version packages are tested against.

Install RStudio

Most people use an integrated development environment (IDE) called RStudio (https://www.rstudio.com/) when they work with R. In fact, once you start using RStudio if you ever have to go back to the regular R environment it feels very punishing. A free version of RStudio is available for most operating systems. The RStudio IDE is also extremely useful for publishing via RMarkdown, package development, and a host of other R-related activities.

R Studio Window

The RStudio window has four main panes. The top left is where you can edit files and highlight code to run. The bottom left is the interactive R session and you can enter commands there directly if you wish. The bottom left also has a Terminal tab so that you can use the command prompt or shell from within the application.



The top right includes a list of all variables in the global environment and also has a History tab of all commands you've run. The bottom right will show

files, packages, plots, and help files while you work. RStudio has many more features which I will not cover.

Packages and Libraries

As I mentioned earlier, packages are collections of code, data, and help files that extend the capabilities of R. Some packages, such as `MASS`, are shipped with R. Others can be downloaded and installed by R using the `install.packages` function (or in RStudio go to Tools > Install Packages). Let's load the `MASS` package and use `fractions` to convert decimals to their nearest rational expression:

```
library("MASS")
fractions(0.333333)
# [1] 1/3
```

Above and throughout this document the output of commands will be displayed following the command as a comment. Comments in R are everything after `#` to the end of the line. Above, the output of the `fractions` command is `1/3`, and so it is displayed as `# [1] 1/3`.

The `MontgomeryDAE` package contains data files from the textbook as well as a copy of this document. Download this package from https://github.com/ehassler/MontgomeryDAE/releases looking for the most current release. A PDF version of this guide will also be attached to the release.

To install a downloaded source package in RStudio you can use `Install Packages` from the `Tools` menu and select `Package Archive File (tar.gz)` from the `Install From` drop-down menu. Then, browse to the location where the `MontgomeryDAE` package was downloaded. If you're using R without RStudio simply use the command

```
install.packages("...path to MontgomeryDAE package...", repos = NULL,
↪  type = "source")
```

to install the package.
Once installed, you can use the help system to see this document as a series of help files:

```
help(package = 'MontgomeryDAE')
```

Follow the link for user guides, package vignettes, and other documentation to see an HTML version of this document for each chapter.

The R Language

If you've worked with a C-like language you'll feel comfortable with R. The syntax of R is something between Javascript and Python. You don't have to worry about declaring types or compiling your code. Instead you can work with R pretty interactively, typing different things into a session and seeing how things work out. This enables an R user to efficiently do exploratory data analysis while preserving the reproducibility of that exploration (since we can simply re-run that set of commands to get the same results).

Everything in R is an object. You can assign functions to variables and return functions from functions. An odd thing about R is that it has functions to interact with the parse tree of the code that's currently being evaluated. Sometimes you'll see R somehow "knows" the name of the variable you passed into a function and uses that as a label on a graph. Don't be alarmed. R is not becoming self-aware.

One example of this oddity is in loading libraries. Previously we loaded the `MASS` package using

```
library("MASS")
```

but we also could have used

```
library(MASS)
```

to load it.

R is an interpreted language with an interactive interpreter. Most of the time I work by writing code, highlighting it, and then have RStudio execute it in the interactive session. I'll then play around with that session to figure out what kind of bugs are occurring or to pull up a help file. R's help system

is very convenient. Once I feel like my analysis is where I want it to be I'll perform a final check before submitting it to a stakeholder: I'll restart my R environment and run the script I've made to make sure the results are what I expect.

To get help with a function or package in R all you have to do is enter a question mark followed by the name. Help files usually include examples as well. Try entering the following:

```
?cat
```

We can use this function to print to the output:

```
cat('hello world')
# hello world
```

We also could have used the `print` function:

```
print("Hello cat!")
# [1] "Hello cat!"
```

We printed the string `'hello world'` and `"Hello cat!"`, each using different quotation marks. This makes no difference, the objects are the same.

```
print(class("Hello"))
# [1] "character"
print(class('Hello'))
# [1] "character"
print('Hello' == "Hello")
# [1] TRUE
```

Arithmetic operators work as expected as well. For example, if we wish to raise 12 to the second power and add one we simply do:

```
print(12^2 + 1)
# [1] 145
```

Assignment to a variable can be done a number of ways, but I try to stick to using `<-`. Let's create a variable and change its value:

```
friends <- 1
friends <- friends + 1
print(friends)
# [1] 2
```

Here we see that we assigned a value of 1 to `friends`, then we added 1 to it, to end up with 2 friends. Use of `=` is also very common. Using that our code would be:

```
friends = 1
friends = friends + 1
print(friends)
# [1] 2
```

An odd thing about R is that you can use a period in variable names. The identifier `total.weight` is perfectly valid, but one should be careful to note that this *is not* referencing the `weight` property on a `total` object. Instead, it's pretty much equivalent to `total_weight`. The use of a period instead of an underscore can be seen lots of places in R because way back in the day underscores were how assignment was performed. It's not anymore so don't worry about it.

There are some important constants in R as well. `TRUE` and `FALSE` mean what you'd expect but must be all capitalized, as does `NULL`. R also has `NA` to represent a not-available value. Let's use the `c` function to concatenate several numbers and then get the median:

```
x <- c(1, 2.2, 3.2e-1)
print(x)
# [1] 1.00 2.20 0.32
print(median(x))
# [1] 1
```

If we add `NA` into our data then the median would also be not available, so:

```
x <- c(1, 2.2, 3.2e-1, NA)
print(median(x))
# [1] NA
```

However, we can tell the median function to just ignore `NA` values:

```
print(median(x, na.rm=TRUE))
# [1] 1
```

Here, I used the name of the argument instead of its position to call the function. This simplifies calling functions with many arguments that have default values.

Functions

Let's create a function to increase our friends. I'll give the second argument a default value so that, if we leave it out of our function call it will take on that value:

```
f <- function(friend.count, new.friends=1){
  return(friend.count + new.friends)
}

friends <- 2
print(f(friends, 2))
# [1] 4
print(f(friends))
# [1] 3
print(f(new.friends=1, friend.count=3))
# [1] 4
```

Note that at the end we reversed the order of the arguments, but since we named them everything acts as we would expect.

In R almost all arguments of a function are passed by value — that is to say a copy is made and then thrown away when the function completes. Consider this example:

```
x <- 1

f <- function(y){
  y <- y + 1
  return(y)
}

print(f(x))
# [1] 2
print(f(x))
# [1] 2
```

One final note about functions. The last expression in a function is returned automatically if no return value is specified. In the interactive session the returned value is sent to `print`. Thus we can rewrite our previous function as follows:

```
x <- 1

f <- function(y){
  y + 1
}

f(x)
# [1] 2
```

Vectors

Most things in R are stored as vectors of the same type. Usually your vector will be numeric, logical (booleans), characters, or a more general object of some kind. R will coerce values to make sure they are the same type within a vector.

```
c(1, 2, 3, '4')
# [1] "1" "2" "3" "4"
```

Above, R decided that character was the best way to store all of those values together.

```
c(1, 2, 3, as.numeric('4'))
# [1] 1 2 3 4
```

Here, `as.numeric` converted the string to a numeric value and so the vector remained as numeric.

Instead of writing the whole sequence of numbers from one to four we could have used R's shortcut notation:

```
1:4
# [1] 1 2 3 4
```

For more complicated sequences we can use the `seq` function.

Vectors can also have names for their elements. This is very helpful in displaying data and accessing it.

```
x <- c('first'=1, 'second'=2, 'third'=3)
x
#  first second  third
#      1      2      3
```

We can then retrieve a specific element either with the numeric offset (starting with 1 for the first element) or by the name

```
print(x[1])
# first
#     1
print(x['second'])
# second
#      2
```

It's worth repeating that indexing in R starts with 1. Many programming languages start with 0, and R does not emit an error when you attempt to access the 0th element.

```
x[0]
# named numeric(0)
```

This is a common source of bugs for in code.

When we compare vectors we get a boolean vector back, and we can use that vector to select elements. Consider the following example of random high temperatures in Phoenix, AZ:

```
x <- c(102, 82, 93, 118, 117, 89, 84, 74, 114, 115)
index <- x < 100
print(index)
#  [1] FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE
x[index]
# [1] 82 93 89 84 74
```

We can also use an integer vector of offsets to select from a vector:

```
index <- 1:(length(x)/2)
print(index)
# [1] 1 2 3 4 5
index <- index * 2
print(index)
# [1]  2  4  6  8 10
print(x[index])
# [1]  82 118  89  74 115
```

Finally, if we provide no index at all we get the full vector back:

```
print(x[])
#  [1] 102  82  93 118 117  89  84  74 114 115
```

Matrices and Arrays

Matrices are essentially two-dimensional vectors, and are a type of array. In R we can define an arbitrary-dimensional vector structure with the `array` function. The methods for accessing elements in a matrix are much the same as the methods we used to access elements of vectors except now we will have commas separating each dimension of the data structure (so matrices have two dimensions, and arrays have one or more dimensions).

First, let's construct a matrix:

```
X <- matrix(c(1,0,0,0,1,0,0,0,1), ncol=3)
X
#      [,1] [,2] [,3]
# [1,]    1    0    0
# [2,]    0    1    0
# [3,]    0    0    1
```

We can use our fancy indices to access sub-matrices:

```
X[c(1,2),2]
# [1] 0 1
```

A three dimensional array can also easily be constructed:

```
W <- array(
  c(
    c(2,1,1,1,2,1,1,1,2),
    c(3,1,1,1,3,1,1,1,3),
    c(4,1,1,1,4,1,1,1,4)
  ),
  dim=c(3,3,3)
)
W
# , , 1
#
#      [,1] [,2] [,3]
# [1,]    2    1    1
# [2,]    1    2    1
# [3,]    1    1    2
#
# , , 2
#
#      [,1] [,2] [,3]
# [1,]    3    1    1
# [2,]    1    3    1
# [3,]    1    1    3
#
# , , 3
#
#      [,1] [,2] [,3]
# [1,]    4    1    1
# [2,]    1    4    1
# [3,]    1    1    4
```

We can access elements of an array in the same way:

```
W[1,1:2,1:2]
#      [,1] [,2]
# [1,]    2    3
# [2,]    1    1
```

As in the case of vectors, if we leave an index blank it returns everything along that dimension. I'll leave the first two places of the index of `W` empty and choose a value for the third.

```
W[ , , 2]
#      [,1] [,2] [,3]
# [1,]    3    1    1
# [2,]    1    3    1
# [3,]    1    1    3
```

Certain operators are expressed between percent symbols. For example, matrix multiplication uses **%*%**.

```
X <- diag(c(1,2,3))
Y <- matrix(c(0,1,0,1,0,0,0,0,1), ncol=3)
X %*% Y
#      [,1] [,2] [,3]
# [1,]    0    1    0
# [2,]    2    0    0
# [3,]    0    0    3
```

Lists

Lists are a less strict association of data. A list doesn't require that the types of entries be the same.

```
y <- list(2, 4, '6', expression(8))
print(y)
# [[1]]
# [1] 2
#
# [[2]]
# [1] 4
#
# [[3]]
# [1] "6"
#
# [[4]]
# expression(8)
```

Note the "key" is given between `[[` and `]]`. This is because we can access an element of a list or get a sublist composed of parts of the original list by using `[` and `]` instead. The following code returns an element:

```
print(y[[1]])
# [1] 2
print(y[[2]])
# [1] 4
```

Whereas the following code return a new list:

```
print(y[1:2])
# [[1]]
# [1] 2
#
# [[2]]
# [1] 4
print(y[1])
# [[1]]
# [1] 2
```

Lists can have string keys as well:

```
friends <- list(
  'Caroline'='Phoenix',
  'John'='California',
  'Tommy'='California',
  'Gina'='Iowa',
  'Adam'='Vancouver',
  'Paul'='Tempe',
  'Mollie'='Utah'
)
print(friends)
# $Caroline
# [1] "Phoenix"
#
# $John
# [1] "California"
#
# $Tommy
# [1] "California"
#
# $Gina
# [1] "Iowa"
#
# $Adam
# [1] "Vancouver"
#
# $Paul
# [1] "Tempe"
#
# $Mollie
# [1] "Utah"
print(friends[['Adam']])
# [1] "Vancouver"
print(friends[c('Adam', 'Caroline')])
# $Adam
# [1] "Vancouver"
#
# $Caroline
# [1] "Phoenix"
```

Note that the key was printed using a `$` when we printed this list. This is another way we can access elements of a list:

```
friends$Paul
# [1] "Tempe"
```

Data Frames

Data frames are the bread-and-butter data structure of R. A `data.frame` is a collection of column vectors that we'll use to feed data into our statistical models:

```
df <- data.frame(
  'time'=c(12.1, 13.3, 13.2, 12.6, 12.8),
  'temp'=c(251, 250, 261, 249, 255)
)
df
#   time temp
# 1 12.1  251
# 2 13.3  250
# 3 13.2  261
# 4 12.6  249
# 5 12.8  255
```

We can access elements of a `data.frame` similar to how we access elements of matrices, for example:

```
df[,'time']
# [1] 12.1 13.3 13.2 12.6 12.8
```

Investigating a Variable

Two very useful commands for figuring out what a variable is are `str` and `dput`. The `str` function displays the structure of a variable:

```
str(df)
# 'data.frame': 5 obs. of  2 variables:
#  $ time: num  12.1 13.3 13.2 12.6 12.8
#  $ temp: num  251 250 261 249 255
```

Here we can see that the `data.frame` has five observations in each of the two columns. Further, both columns are numeric.

The `dput` function writes an ASCII text representation of the object (i.e. we can paste `dput`'s output into R and be guaranteed to get the same object back):

```
dput(df)
# structure(list(time = c(12.1, 13.3, 13.2, 12.6, 12.8), temp = c(251,
↪
# 250, 261, 249, 255)), class = "data.frame", row.names = c(NA,
# -5L))
```

We can show their equivalence:

```
df2 <- structure(list(time = c(12.1, 13.3, 13.2, 12.6, 12.8), temp =
↪  c(251,
250, 261, 249, 255)), class = "data.frame", row.names = c(NA,
-5L))
df==df2
#      time temp
# [1,] TRUE TRUE
# [2,] TRUE TRUE
# [3,] TRUE TRUE
# [4,] TRUE TRUE
# [5,] TRUE TRUE
```

You'll note that `dput` told us that under the hood the `data.frame` is a list, and indeed we can access it like a list:

```
df$time
# [1] 12.1 13.3 13.2 12.6 12.8
```

Factors

Sometimes we want to indicate that a column of a `data.frame` represents qualitative information. For example, if we had assigned people to groups then it doesn't make sense for group 2 to be twice as much a group as group 1 and group 4 to be twice as much a group as group 2. In R we handle this with factors.

```
factor(c(1,2,3,1,3,2,3,3,2,1,2,2,3))
#  [1] 1 2 3 1 3 2 3 3 2 1 2 2 3
# Levels: 1 2 3
```

Some functions understand that levels represent some qualitative information. Let's load and plot the canonical Iris data set that describes 50 of each of three species of iris flower in terms of the petal length and width and the sepal length and width (see https://en.wikipedia.org/wiki/Iris_flower_data_set for more about the data set. You can find the data set in the `datasets` package as `iris` or `iris3`.)

```
library(datasets)
data(iris)
```

This loaded the iris data set into a `data.frame` named `iris`. To verify this we can use `str` to examine the structure of the `iris` object:

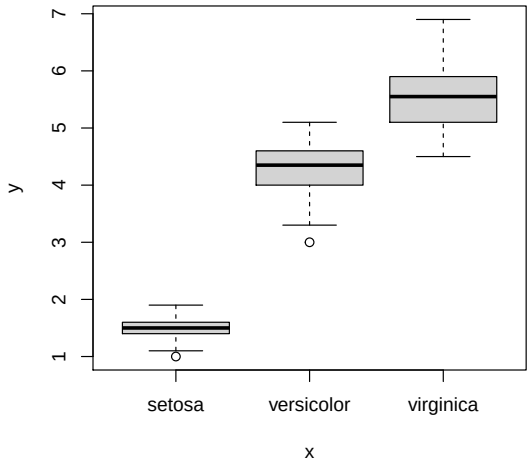
```
str(iris)
# 'data.frame': 150 obs. of  5 variables:
#  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
#  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
#  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
#  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
#  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1
↪  1 1 1 1 1 1 ...
```

We can get summary statistics on each column of the `data.frame` as follows:

```
summary(iris)
#   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
#  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
#  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
#  Median :5.800   Median :3.000   Median :4.350   Median :1.300
#  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
#  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
#  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
#        Species
#  setosa    :50
#  versicolor:50
#  virginica :50
#
#
#
```

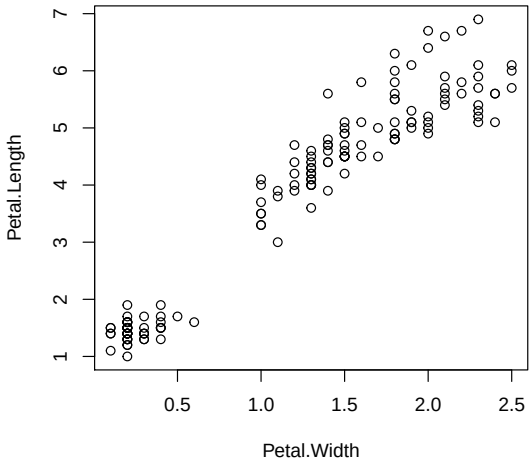
We see there are 3 kinds of iris. Let's plot `Petal.Length` in terms of `Species`.

```
species <- iris[,'Species']
petalLength <- iris[,'Petal.Length']
plot(species, petalLength)
```



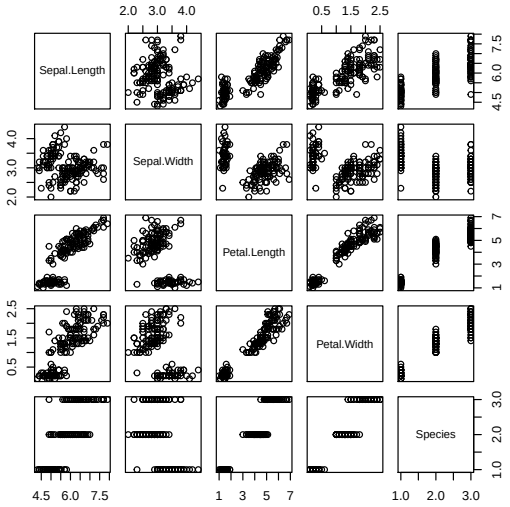
R decided that, since we're using a factor for x, a boxplot is most appropriate for showing the data. Now let's plot `Petal.Length` by `Petal.Width`:

```
plot(iris[,c('Petal.Width','Petal.Length')])
```



We get a totally different *kind* of plot. This scatter plot relates the petal width to the petal length in a way we'd expect. Let's go hog wild!

```
plot(iris)
```



This produces a lattice plot that we can use to see all kinds of two way relationships. You may be asking, "Why does `Species` have levels '1.0', '2.0', and '3.0'?" Well, those are the indices of the levels of the species factor:

```
levels(iris$Species)
# [1] "setosa"     "versicolor" "virginica"
```

If we were to convert the factor to numeric we would get back those offsets:

```
as.numeric(iris$Species)
#   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
↪  1 1 1 1
#  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
↪  2 2 2 2
#  [71] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3
↪  3 3 3 3
# [106] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
↪  3 3 3 3
# [141] 3 3 3 3 3 3 3 3 3 3
```

If we wanted to get the original strings back we could do

```
as.character(iris$Species)
```

Formula

Another odd thing about R (the third odd thing for those keeping count) is formula objects. Let's say we want to perform a linear regression to predict petal length using petal width and species. R uses something called *Wilkinson's notation* to describe this relationship. The following is an example formula relating `Petal.Length` to `Petal.Width` and `Species`.

```
formula <- Petal.Length ~ Petal.Width + Species
print(class(formula))
# [1] "formula"
print(formula)
# Petal.Length ~ Petal.Width + Species
```

We can use the formula with the `lm` function to get the least squares fit:

```
lm(Petal.Length ~ Petal.Width + Species, data=iris)
#
# Call:
# lm(formula = Petal.Length ~ Petal.Width + Species, data = iris)
#
# Coefficients:
# (Intercept)  Petal.Width     Species1     Species2
#       2.536        1.019       -1.325        0.373
```

We can also pass a string to `lm` to get the same results:

```
lm("Petal.Length ~ Petal.Width + Species", data=iris)
#
# Call:
# lm(formula = "Petal.Length ~ Petal.Width + Species", data = iris)
#
# Coefficients:
# (Intercept)  Petal.Width     Species1     Species2
#       2.536        1.019       -1.325        0.373
```

The formula also describes how the raw data is to be transformed for modeling. If we used `*` instead of `+` then R would assume we want `Petal.Width`, `Species`, and the two-factor interaction term `Species:Petal.Width`:

```
lm(Petal.Length ~ Petal.Width * Species, data=iris)
#
# Call:
```

```
# lm(formula = Petal.Length ~ Petal.Width * Species, data = iris)
#
# Coefficients:
#          (Intercept)           Petal.Width              Species1
#               2.4498                1.0210               -1.1223
#              Species2  Petal.Width:Species1  Petal.Width:Species2
#              -0.6686               -0.4745                0.8483
```

We can do a lot of fancy things using Wilkinson's notation and it's becoming common to see it used in Python (`patsy` package) and Matlab. I'll refer you to a short guide for R from Richard Hahn and Matlab's documentation for more information about the notation.

Where to Learn More

There are a lot of resources you can use to learn more about R.

- A (Very) Short Introduction to R by Torfs and Braur is an excellent concise introductory PDF.
- The book R In Action and website Quick-R by Rob Kabacoff are also excellent resources.
- DataCamp has a free introduction to R course.
- Though much older, the book "Modern Applied Statistics With S-Plus" by Venables and Ripley is a great go-to resource for doing statistical analysis in S. This often explains some of the more arcane details of how to use R itself.
- The R community and user groups remind me of a friendly data cult: my experience has been of people who are passionate and go out of their way to be helpful to people learning R. Perform a web search to find an R user group or meetup in your area.

Chapter 2

Simple Comparative Experiments

Chapter 2 begins with several methods of visualization that are easily reproducible within R. These visualizations are based on the data in Table 2.1 which records the results of an experiment where a researcher was comparing the strength of a new mortar formulation to that of the original unmodified formulation.

ModifiedMortar	UnmodifiedMortar
16.85	16.62
16.40	16.75
17.21	17.37
16.35	17.12
16.52	16.98
17.04	16.87
16.96	17.34
17.15	17.02
16.59	17.08
16.57	17.27

In R, the object `Table2.1` is a `data.frame` that has a column `ModifiedMortar` and a column `UnmodifiedMortar`. The structure of this object is given below.

```
str(Table2.1)
# 'data.frame': 10 obs. of  2 variables:
#  $ ModifiedMortar  : num  16.9 16.4 17.2 16.4 16.5 ...
#  $ UnmodifiedMortar: num  16.6 16.8 17.4 17.1 17 ...
```

We can use the R command `attach` to take the columns of the `data.frame` and make them accessible by name (e.g. we can use `ModifiedMortar` instead of `Table2.1$ModifiedMortar`):

```
attach(Table2.1)
all(ModifiedMortar == Table2.1$ModifiedMortar)
# [1] TRUE
```

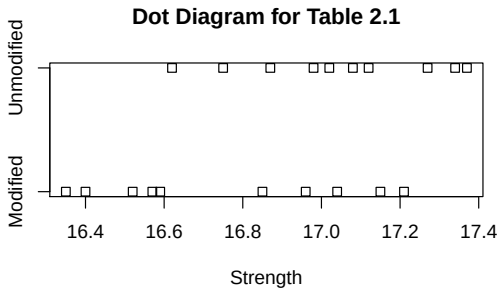
When we're all done with the data we should call `detach(Table2.1)`.

With the data loaded let's move on to some of the visualizations in Chapter 2.

Visualizations

We begin with the dot plot of Figure 2.1:

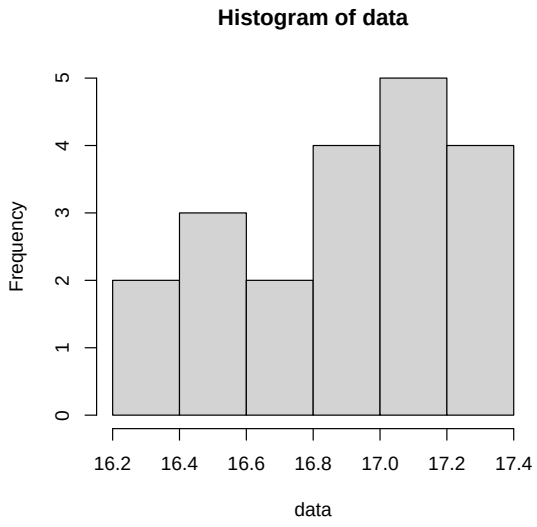
```
stripchart(
    list(
        'Modified'=ModifiedMortar,
        'Unmodified'=UnmodifiedMortar
    ),
    main="Dot Diagram for Table 2.1",
    xlab='Strength'
)
```



The graph seems to suggest that the modified formulation has a lower average strength than that of the unmodified formulation.

Figure 2.2 in the text is a histogram of 200 observations of metal recovery for which no data is given. Though we lack the full data set, let's construct a histogram of the 20 data points from combining the modified and unmodified mortar formulation data:

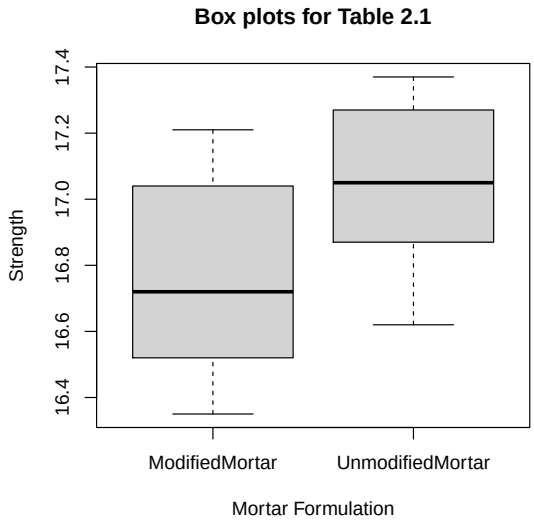
```
data <- c(ModifiedMortar, UnmodifiedMortar)
hist(data)
```



Though it was easy to construct, the histogram is not giving us a lot of information in this case due to the fact that we have such a small amount of data.

The dot plot of Figure 2.1 can become very difficult to interpret when the number of observations becomes large. In these cases we can use a box plot which is constructed based on summary statistics of the data. The box plots of Figure 2.3 are presented next.

```
bp.returned <- boxplot(
    Table2.1,
    main="Box plots for Table 2.1",
    xlab='Mortar Formulation',
    ylab='Strength'
)
```



This plot also suggests that the modified mortar formulation has a lower mean strength than the unmodified formulation. Note that I assigned the return value from the `boxplot` function to a variable `bp.returned`. Several graphics functions return data used in the graphical construction. The `boxplot` function returns the summary statistics used to create the plots:

```
print(bp.returned)
# $stats
#       [,1]  [,2]
# [1,] 16.35 16.62
# [2,] 16.52 16.87
# [3,] 16.72 17.05
# [4,] 17.04 17.27
# [5,] 17.21 17.37
#
# $n
# [1] 10 10
#
# $conf
#          [,1]     [,2]
# [1,] 16.46019 16.85014
# [2,] 16.97981 17.24986
#
# $out
# numeric(0)
#
# $group
# numeric(0)
#
# $names
# [1] "ModifiedMortar"   "UnmodifiedMortar"
```

Statistics and Distributions

The box plot was a nice way to illustrate the distribution of our data, but we could also use the `summary` function to get summary statistics:

```
summary(Table2.1)
#  ModifiedMortar  UnmodifiedMortar
#  Min.   :16.35   Min.   :16.62
#  1st Qu.:16.53   1st Qu.:16.90
#  Median :16.72   Median :17.05
#  Mean   :16.76   Mean   :17.04
#  3rd Qu.:17.02   3rd Qu.:17.23
#  Max.   :17.21   Max.   :17.37
```

Other statistics of interest, such as standard deviation (or variance) and empirical quantiles, are easy to collect. I'll define a new `data.frame` with some statistics:

```
dstat <- data.frame(
    'Modified'=c(
        mean(ModifiedMortar),
        var(ModifiedMortar),
        sd(ModifiedMortar),
        length(ModifiedMortar)
    ),
    'Unmodified'=c(
        mean(UnmodifiedMortar),
        var(UnmodifiedMortar),
        sd(UnmodifiedMortar),
        length(UnmodifiedMortar)
    ),
    row.names=c('mean', 'variance', 'sd','obs')
)
```

The `dstat` object then contains the following:

	Modified	Unmodified
mean	16.7640000	17.0420000
variance	0.1001378	0.0614622
sd	0.3164455	0.2479158
obs	10.0000000	10.0000000

If we want to create a pooled estimator of variance S_p^2 we can do so manually:

```
n1 <- dstat['obs','Modified']
n2 <- dstat['obs','Unmodified']
S1sq <- dstat['var','Modified']
S2sq <- dstat['var','Unmodified']
pooled.var <- ((n1 - 1) * S1sq + (n2 - 1) * S2sq) / (n1 + n2 - 2)
print(pooled.var)
# [1] 0.0808
```

This calculation matches that of the text. Similarly, we can create the t-statistic to test the hypotheses

$$H_0 : \mu_{\text{Modified}} = \mu_{\text{Unmodified}} \quad \text{vs.} \quad H_1 : \mu_{\text{Modified}} \neq \mu_{\text{Unmodified}} \ ,$$

```
ybar1 <- dstat['mean', 'Modified']
ybar2 <- dstat['mean', 'Unmodified']
t0 <- (ybar1 - ybar2) / sqrt(pooled.var * (1/n1 + 1/n2))
print(t0)
# [1] -2.186876
```

We can evaluate the statistic in two ways. First, we can use the critical region approach and find $t_{0.025,18}$ using the `qt` function. Note that in R, all of the quantile functions give the area to the left (the lower area), but this text and many others take the symbol $t_{\alpha,n}$ to mean the critical point where the area *to the right* is α. To adjust for this we give the argument `lower.tail=FALSE`.

```
critical.t <- qt(0.025, 18, lower.tail=FALSE)
print(critical.t)
# [1] 2.100922
```

Because $|t_0| > t_{0.025,18}$ we would reject H_0. The other option is to find the p-value using the pt function:

```
2 * (1 - pt(abs(t0), 18))
# [1] 0.04219672
```

Since the p-value was less that 0.05 we would again reject H_0.

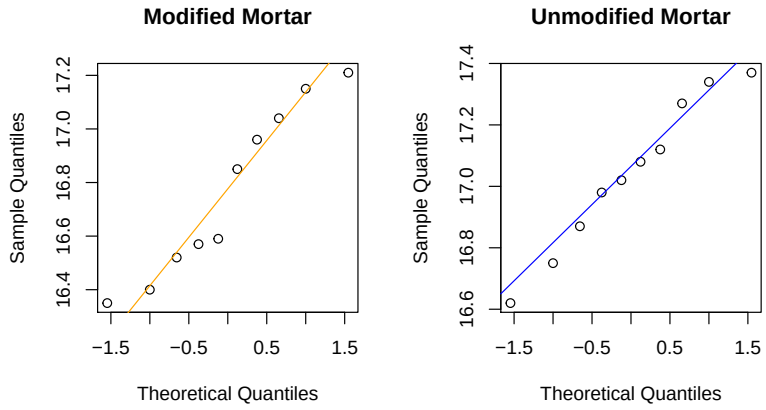
That was a lot of typing to just perform a two-sample t-test! A shorter path would be to use R's `t.test` function. This requires us to reshape our data into a column for strength and a column that indicates if the observation was for the modified or unmodified formulation:

```
df <- data.frame(
    'Strength'=c(ModifiedMortar, UnmodifiedMortar),
    'Mortar'=factor(c(rep('Modified', 10), rep('Unmodified', 10)))
)
t.test(Strength ~ Mortar, data=df, paired=FALSE, var.equal=TRUE)
#
#   Two Sample t-test
#
# data:  Strength by Mortar
# t = -2.1869, df = 18, p-value = 0.0422
# alternative hypothesis: true difference in means is not equal to 0
# 95 percent confidence interval:
#  -0.54507339 -0.01092661
# sample estimates:
#   mean in group Modified mean in group Unmodified
#                   16.764                   17.042
```

The `t.test` function is giving us a lot of information: the statistic, the degrees of freedom, the p-value, a confidence interval at 95%, and estimated means for the modified and unmodified mortars.

Section 2.4.3 includes the normal probability plots of tension bond strength in Figure 2.11. It's easiest to construct separate plots for each data set as follows:

```
op <- par(mfrow=c(1,2))
qqnorm(ModifiedMortar, main="Modified Mortar")
qqline(ModifiedMortar, col='orange')
qqnorm(UnmodifiedMortar, main="Unmodified Mortar")
qqline(UnmodifiedMortar, col='blue')
```




```
par(op)
```

Visual inspection doesn't suggest any strong violation of the assumption. Note that `par` is setting the graphical parameter `mfrow` so that we can plot two graphs next to each other. When setting with `par` the original value is returned and stored in `op`, which we then reset at the end.

```
detach(Table2.1)  # Clean up since we're done with this data.
```

Power and Sample Size

For power and sample size calculations we'll use the `pwr` package (Champley et al. 2018). For the t-test we simply leave out the argument we want calculated. Recall that

$$d = \frac{|\mu_1 - \mu_2|}{\sigma}$$

so for the mortar experiment we'd want $d = 2$, then if each group has 10 observations we find the power:

```
library(pwr)
pwr.t.test(n=10, d=2, sig.level=0.05, type='two.sample',
↪  alternative='two.sided')
#
#      Two-sample t test power calculation
#
#               n = 10
```

```
#               d = 2
#       sig.level = 0.05
#           power = 0.988179
#     alternative = two.sided
#
# NOTE: n is number in *each* group
```

Here, our power was 98.8%. How many observations would we have needed to get the power of our test to 90%?

```
pwr.t.test(power=0.9, d=2, sig.level=0.05, type='two.sample',
↪  alternative='two.sided')
#
#      Two-sample t test power calculation
#
#               n = 6.386753
#               d = 2
#       sig.level = 0.05
#           power = 0.9
#     alternative = two.sided
#
# NOTE: n is number in *each* group
```

If we were planning the mortar experiment we'd want to make sure we had about 7 observations of each mortar to achieve approximately 90% power.

Unequal Variance — Example 2.1

In Example 2.1 a study of the normalized fluorescence (after two hours) was undertaken for nerve tissue from 12 mice and muscle tissue from 12 mice. Note that our data is not paired. We wish to test

$$H_0 : \mu_{\text{Nerve}} = \mu_{\text{Muscle}} \quad \text{vs.} \quad H_A : \mu_{\text{Nerve}} > \mu_{\text{Muscle}} \ .$$

The data from this experiment is presented in Table 2.3.

Observation	Nerve	Muscle
1	6625	3900
2	6000	3500
3	5450	3450
4	5200	3200
5	5175	2980
6	4900	2800
7	4750	2500
8	4500	2400
9	3985	2200
10	900	1200
11	450	1150
12	2800	1130

Summary statistics for the data are presented below:

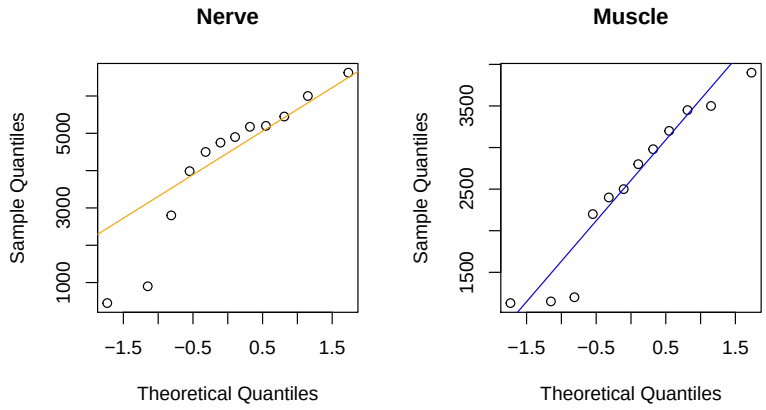
```
summary(Table2.3[,c('Nerve','Muscle')])
#      Nerve          Muscle
#  Min.   : 450   Min.   :1130
#  1st Qu.:3689   1st Qu.:1950
#  Median :4825   Median :2650
#  Mean   :4228   Mean   :2534
#  3rd Qu.:5262   3rd Qu.:3262
#  Max.   :6625   Max.   :3900
```

The summary statistics left out the standard deviations, which we can calculate below:

```
c('Nerve'=sd(Table2.3$Nerve), 'Muscle'=sd(Table2.3$Muscle))
#     Nerve    Muscle
# 1917.9919  960.5061
```

Those estimates are strikingly different. Examining the normal probability plots we find a very distinct difference in slope, suggesting that the variances are unequal.

```
op <- par(mfrow=c(1,2))
qqnorm(Table2.3$Nerve, main="Nerve")
qqline(Table2.3$Nerve, col='orange')
qqnorm(Table2.3$Muscle, main="Muscle")
qqline(Table2.3$Muscle, col='blue')
```




```
par(op)
```

To test our hypothesis we again use the `t.test` procedure. To do the one-sided alternative test we use the argument `alternative='greater'`. Note that the order of the factor `Tissue` is *Muscle* then *Nerve*, so `greater` is testing that Nerve''s mean is greater than Muscle's mean.

```
df <- data.frame(
    'Flourescence'=c(Table2.3$Muscle, Table2.3$Nerve),
    'Tissue'=factor(c(rep('Muscle', 12), rep('Nerve', 12)))
)
t.test(Flourescence ~ Tissue, data=df, paired=FALSE, var.equal=FALSE,
↪  alternative='greater')
#
#   Welch Two Sample t-test
#
# data:  Flourescence by Tissue
# t = -2.7353, df = 16.191, p-value = 0.9927
# alternative hypothesis: true difference in means is greater than 0
# 95 percent confidence interval:
#  -2774.063       Inf
# sample estimates:
# mean in group Muscle  mean in group Nerve
#             2534.167             4227.917
```

Paired Comparisons

In Section 2.5.1 the hardness testing experiment is introduced. In this experiment, the depth of two different tips in samples of metal is measured. Note

that each tip was tested on the same piece of metal, so our data is paired. The data for this experiment is given in Table 2.6.

Specimen	Tip1	Tip2
1	7	6
2	3	3
3	3	5
4	4	3
5	8	8
6	3	2
7	2	4
8	9	9
9	5	4
10	4	5

To perform the paired t-test we do the following:

```
df <- data.frame(
    'Depth'=c(Table2.6$Tip1, Table2.6$Tip2),
    'Tip'=factor(c(rep('Tip1', 10), rep('Tip2', 10)))
)
t.test(Depth ~ Tip, data=df, paired=TRUE)
#
#   Paired t-test
#
# data:  Depth by Tip
# t = -0.26414, df = 9, p-value = 0.7976
# alternative hypothesis: true difference in means is not equal to 0
# 95 percent confidence interval:
#  -0.9564389  0.7564389
# sample estimates:
# mean of the differences
#                    -0.1
```

With a p-value of 0.8 we certainly don't have evidence to reject our null hypothesis that the means are equal.

Notes

There are many other functions in R that handle a lot of the analysis we might want to do. A non-exhaustive list follows:

- `binom.test` - Exact binomial test.
- `chisq.test` - Pearson's χ^2-test for count data.
- `krustkal.test` - Kruskal-Wallis rank-sum test.
- `prop.test` - A one or two sample test of proportions.
- `quantile` - Get the empirical quantiles of a set of data.
- `shapiro.test` - Tests for normality of the data.
- `summary` - Produce a summary of data or other object.
- `t.test` - The two sample t-test.
- `table` - Tabulate data. Note that if you `summary` a table then a χ^2 test of independence is performed.
- `var.test` - Conduct the two sample F-test for population variance
- `wilcox.test` - Test and construct confidence interval for the median.

References

Champley, Stephane, Claus Ekstrom, Peter Dalgaard, Jeffrey Gill, Stephan Weibelzahl, Aditya Anandkumar, Clay Ford, Rovert Volcic, and Helios De Rosario. 2018. *Pwr: Basic Functions for Power Analysis*. https://cran.r-project.org/package=pwr.

Chapter 3

Experiments with a Single Factor: The Analysis of Variance

When factors are used in linear models then there's some flexibility in how we input the factor into the model matrix. By default, R uses a function `contr.treatment` to encode unordered factors and `contr.poly` to encode ordered factors. If a factor has k levels then `contr.treatment` function will encode the last $k-1$ as indicator functions taking values 0 when the level is not present and 1 when the level is present. This causes the first level of the factor to be aliased into the intercept term in the model and the remaining coefficients represent the difference from this mean. If we want the levels of the factor to be encoded in such a way that the sum of their coefficients is zero we have to use `contr.sum`. The following will set the default for unordered factor to be `contr.sum` and leave the ordered factor behavior as the default.

```
options(contrasts=c('unordered'='contr.sum', 'ordered'='contr.poly'))
```

Note that this setting only affects the effect estimates — things like ANOVA table sums of squares shouldn't be affected.

Example 3.1 — The Plasma Etching Experiment

The plasma etching experiment of Table 3.1 relates the RF power to the depth of the etching. At each of the four power levels there were 5 observations. The structure of the `data.frame` is given below:

```
str(Table3.1)
# 'data.frame': 20 obs. of  2 variables:
#  $ Power      : chr  "160" "160" "160" "160" ...
#  $ Observation: num  575 542 530 539 570 565 593 590 579 610 ...
```

Fitting the ANOVA model we find that we can reject the null hypothesis that all of the power levels have the same average etch amount:

```
summary(aov(Observation ~ Power, data=Table3.1))
#             Df Sum Sq Mean Sq F value   Pr(>F)
# Power        3  66871   22290    66.8 2.88e-09 ***
# Residuals   16   5339     334
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Compare this result to Table 3.4 in Example 3.1. To get the effect estimates in Example 3.3 we have to do a little extra work:

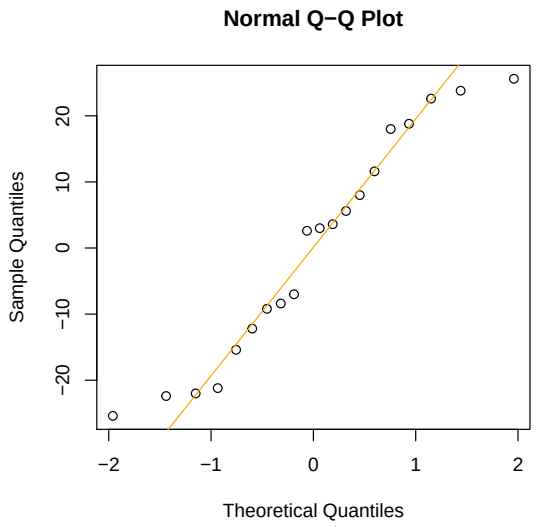
```
model <- lm(Observation ~ Power, data=Table3.1)
summary(model)
#
# Call:
# lm(formula = Observation ~ Power, data = Table3.1)
#
# Residuals:
#    Min     1Q Median     3Q    Max
#  -25.4  -13.0    2.8   13.2   25.6
#
# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept)  617.750      4.085 151.234  < 2e-16 ***
# Power1       -66.550      7.075  -9.406 6.39e-08 ***
# Power2       -30.350      7.075  -4.290 0.000563 ***
# Power3         7.650      7.075   1.081 0.295602
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
```

```
# Residual standard error: 18.27 on 16 degrees of freedom
# Multiple R-squared:  0.9261,  Adjusted R-squared:  0.9122
# F-statistic:  66.8 on 3 and 16 DF,  p-value: 2.883e-09
```

Since the sum of the coefficients for the `Power` factor is zero we know the `Power4` coefficient is 89.25.

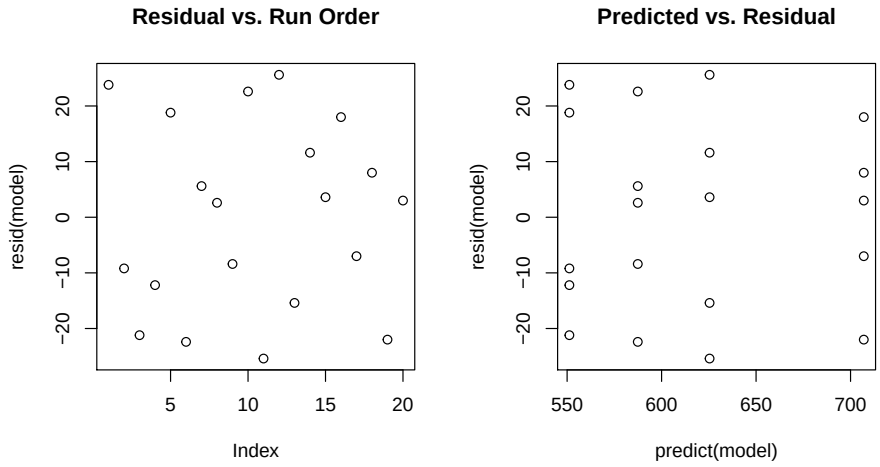
To check the model adequacy we can generate the normal probability plot of the residuals from the model.

```
qqnorm(resid(model))
qqline(resid(model), col='orange')
```



The normal probability plot doesn't show any strong signs of non-normality. Next, let's examine the residual-by-run-order plot to look for patterns that would indicate a lack of independence and the predicted-versus-residual plot to check for equality of variance:

```
op <- par(mfrow=c(1, 2))
plot(resid(model), main='Residual vs. Run Order')
plot(x=predict(model), y=resid(model), main='Predicted vs. Residual')
```




```
par(op)
```

Neither plot seems to show strong signs of trouble. We can also perform a formal test for equality of variance. Bartlett's test, described in Section 3.4, tests

$$H_0 : \sigma_1^2 = \sigma_2^2 = \ldots = \sigma_a^2 \quad \text{vs.} \quad H_A : \text{above not true for at leas one } \sigma_i^2 \ .$$

This is easy to do in R:

```
bartlett.test(Observation ~ Power, data=Table3.1)
#
#   Bartlett test of homogeneity of variances
#
# data:  Observation by Power
# Bartlett's K-squared = 0.43349, df = 3, p-value = 0.9332
```

Bartlett's test fails to reject the null hypothesis. We find no strong evidence that the model is an inadequate description of the data.

Example 3.5 and Variance Stabilizing Transforms

In Example 3.5 an engineer is investigating if there's a difference between the four methods of estimating flood flow frequency. Note that here, estimation

method is coded as 1, 2, 3, or 4. It's important to check that these are a factor so that R doesn't try to treat *4* as twice the estimation method as *2*:

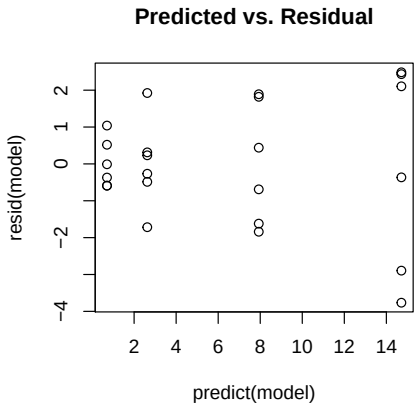
```
str(Example3.5)
# 'data.frame': 24 obs. of  2 variables:
#  $ EstimationMethod: chr  "1" "1" "1" "1" ...
#  $ Observation     : num  0.34 0.12 1.23 0.7 1.75 0.12 0.91 2.94
↪  2.14 2.36 ...
```

Having verified `EstimationMethod` is a factor, let's construct the ANOVA table as in Table 3.8:

```
model <- aov(Observation ~ EstimationMethod, data=Example3.5)
summary(model)
#                  Df Sum Sq Mean Sq F value   Pr(>F)
# EstimationMethod  3  708.3   236.1   76.07 4.11e-11 ***
# Residuals        20   62.1     3.1
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The following diagnostic plot of predicted-versus-residual suggests that the model is not appropriate:

```
plot(predict(model), resid(model), main='Predicted vs. Residual')
```

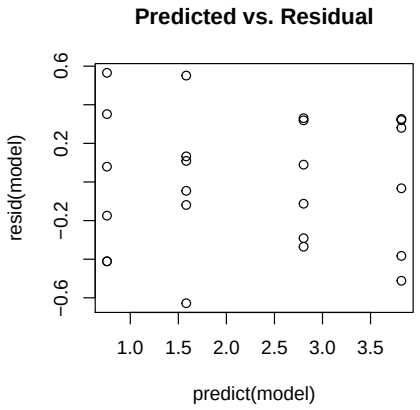


As suggested in the text, we apply a $y^\star = \sqrt{y}$ transform and refit the model, achieving Table 3.10 in the text:

```
y <- sqrt(Example3.5$Observation)
model <- aov(y ~ EstimationMethod, data=Example3.5)
summary(model)
#                  Df Sum Sq Mean Sq F value  Pr(>F)
# EstimationMethod  3  32.68  10.895   81.05 2.3e-11 ***
# Residuals        20   2.69   0.134
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The new predicted versus residual plot looks much better:

```
plot(predict(model), resid(model), main='Predicted vs. Residual')
```



Even though the conclusions drawn from the untransformed data are the same as with the transformed data, having validated our model assumptions means we can put more trust in our results.

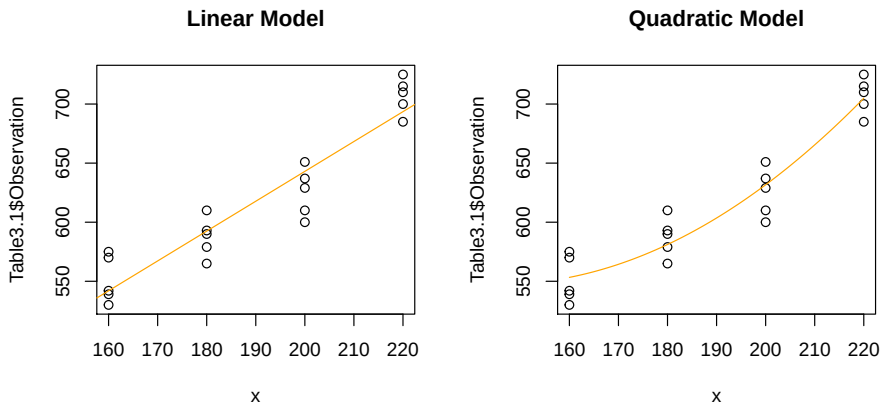
Regression Model

Section 3.5.1 suggests using a regression model to fit the etch data of Table 3.1. To do this, first we convert `Power` to a numeric vector `x`. Next, we fit the linear and quadratic models. The symbol `I(x^2)` tells R that `x^2` is how you want to transform `x` and include it in the model. If we left the `I` part off the result would be that R would effectively treat it as a second `x` term in the formula and ignore it.

```
x <- as.numeric(as.character(Table3.1$Power))
linear.model <- lm(Observation ~ x, data=Table3.1)
quadratic.model <- lm(Observation ~ x + I(x^2), data=Table3.1)
```

Next let's plot the linear versus the quadratic fits:

```
op <- par(mfrow=c(1,2))
plot(x, Table3.1$Observation, main='Linear Model')
abline(a=coef(linear.model)[1], b=coef(linear.model)[2], col='orange')
plot(x, Table3.1$Observation, main='Quadratic Model')
u <- seq(from=160, to=220, length.out=100)
v <- cbind(rep(1, length(u)), u, u^2) %*% coef(quadratic.model)
lines(u, v, col='orange')
```




```
par(op)
```

Visual inspection suggests that the quadratic model is a better fit for the data. This model lets us estimate mean etch depth at power levels between 160 and 220, which would not be possible using the original ANOVA model.

Contrasts

Recall the ANOVA model for the plasma etching experiment:

```
model <- aov(Observation ~ Power, data=Table3.1)
summary(model)
#             Df Sum Sq Mean Sq F value   Pr(>F)
# Power        3  66871   22290    66.8 2.88e-09 ***
```

```
# Residuals   16   5339     334
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

One way to calculate individual contrasts is to use the `emmeans` package (Lenth et al. 2019). Here we'll create an object and then derive the contrasts from it:

```
library(emmeans)
emmodel <- emmeans(model,   ~ Power)
contrast(emmodel, list(
    'C1'=c(1, -1, 0, 0),
    'C2'=c(1, 1, -1, -1),
    'C3'=c(0, 0, 1, -1)
))
#  contrast estimate   SE df t.ratio p.value
#  C1          -36.2 11.6 16  -3.133 0.0064
#  C2         -193.8 16.3 16 -11.861 <.0001
#  C3          -81.6 11.6 16  -7.063 <.0001
```

These contrasts match those given in Example 3.6. We can also do all pairwise comparisons of the `Power` factor using the Tukey adjustment of Example 3.7:

```
pairs(emmodel, adjust="tukey")
#  contrast  estimate   SE df t.ratio p.value
#  160 - 180    -36.2 11.6 16  -3.133 0.0294
#  160 - 200    -74.2 11.6 16  -6.422 <.0001
#  160 - 220   -155.8 11.6 16 -13.485 <.0001
#  180 - 200    -38.0 11.6 16  -3.289 0.0216
#  180 - 220   -119.6 11.6 16 -10.352 <.0001
#  200 - 220    -81.6 11.6 16  -7.063 <.0001
#
# P value adjustment: tukey method for comparing a family of 4
↪  estimates
```

Both Fisher's LSD and Dunnett's test can be run by comparing the derived critical value to the estimate given above.

Determining Sample Size

As in Chapter 2 we will use the `pwr` package to calculate power/sample size. For the single factor ANOVA, we wish to calculate an effect size f to use in our power calculation. Let there be a levels each with n samples for a total

of $N = an$ samples overall. Let μ_i be the mean of level i, μ the overall mean (mean of all means), and let σ be the standard deviation. The effect size f is

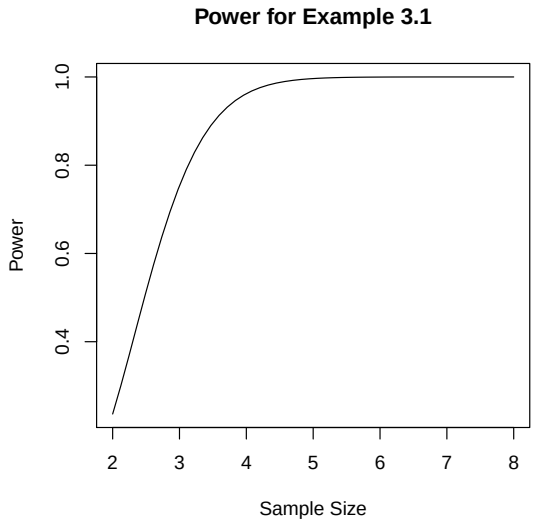
$$f = \frac{1}{\sigma}\sqrt{\frac{1}{a}\sum_{i=1}^{k}(\mu_i - \mu)^2}$$

In Example 3.1, the standard deviation is 25 with prospective means 575, 600, 650, and 675. Thus, if we have three samples at each mean and our test is to have $\alpha = 0.01$ then the power would be:

```
library(pwr)
w <- c(575, 600, 650, 675)
a <- length(w)
sigma <- 25
f <- sqrt((1/a)* sum((w - mean(w))^2)) / sigma
pwr.anova.test(k=4, f=f, n=3, sig.level=0.01)
#
#      Balanced one-way analysis of variance power calculation
#
#               k = 4
#               n = 3
#               f = 1.581139
#       sig.level = 0.01
#           power = 0.7534019
#
# NOTE: n is number in each group
```

so about 75%. We can construct the power curve easily:

```
library(pwr)
n <- seq(from=2, to=8, length.out=50)
y <- pwr.anova.test(k=4, f=f, sig.level=0.01, n=n)$power
plot(n, y, type='l',
     main='Power for Example 3.1',
     xlab='Sample Size',
     ylab='Power')
```



According to the plot, to get 90% power would require a sample size of 4 at each group (16 overall). If we wish for 90% power we can get that directly:

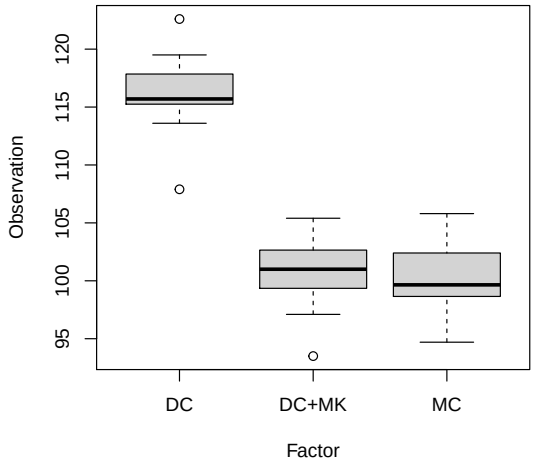
```
pwr.anova.test(k=4, f=f, sig.level=0.01, power=0.9)
#
#      Balanced one-way analysis of variance power calculation
#
#               k = 4
#               n = 3.520261
#               f = 1.581139
#       sig.level = 0.01
#           power = 0.9
#
# NOTE: n is number in each group
```

Rounding up, we'd require $n = 4$ samples at each level of `Power`.

Chocolate Consumption Experiment of Section 3.8.1

Consider the experiment described in Section 3.8.1 where three different types of chocolate were given to subjects and then the antioxidant capacity of their blood plasma was measured. A boxplot of the data is presented below:

```r
boxplot(Observation ~ Factor, data=Table3.12)
```



Next, we fit an ANOVA model and look for the general significance test:

```r
summary(aov(Observation ~ Factor, data=Table3.12))
#             Df Sum Sq Mean Sq F value   Pr(>F)
# Factor       2 1952.6   976.3   93.58 2.52e-14 ***
# Residuals   33  344.3    10.4
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This indicates that the chocolate concoctions had differing mean effects on antioxidant levels. To estimate the individual effects we use a linear model without an intercept:

```r
means.model <- lm(Observation ~ Factor - 1, data=Table3.12)
print(summary(means.model))
#
# Call:
# lm(formula = Observation ~ Factor - 1, data = Table3.12)
#
# Residuals:
#     Min      1Q  Median      3Q     Max
# -8.1583 -1.4083 -0.3583  1.9375  6.5417
#
# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# FactorDC    116.0583     0.9324   124.5   <2e-16 ***
# FactorDC+MK 100.7000     0.9324   108.0   <2e-16 ***
# FactorMC    100.1833     0.9324   107.4   <2e-16 ***
# ---
```

```
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 3.23 on 33 degrees of freedom
# Multiple R-squared:  0.9991,  Adjusted R-squared:  0.9991
# F-statistic: 1.29e+04 on 3 and 33 DF,  p-value: < 2.2e-16
```

Next, we generate confidence intervals for each chocolate level:

```
confint(means.model)
#                 2.5 %   97.5 %
# FactorDC    114.16125 117.9554
# FactorDC+MK  98.80292 102.5971
# FactorMC     98.28625 102.0804
```

Next, we'll look at all pairwise comparisons (without making any adjustment for multiple comparisons):

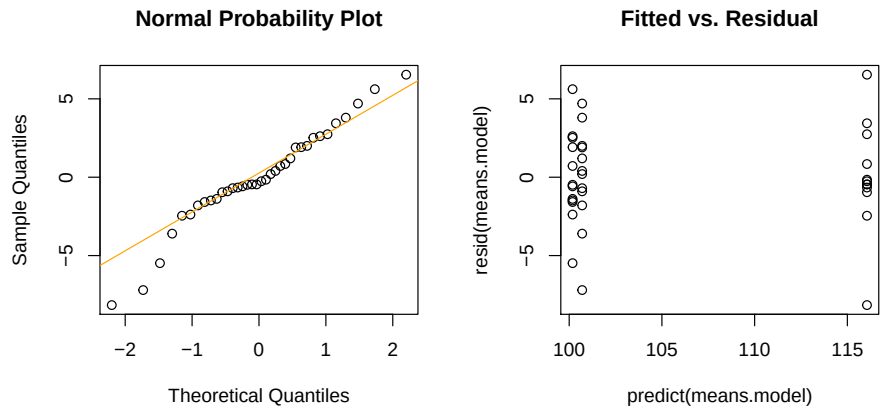
```
pairwise.model <- pairs(emmeans(aov(Observation ~ Factor,
↪  data=Table3.12), ~Factor), adjust='none')
print(pairwise.model)
#  contrast     estimate   SE df t.ratio p.value
#  DC - (DC+MK)   15.358 1.32 33 11.647  <.0001
#  DC - MC        15.875 1.32 33 12.039  <.0001
#  (DC+MK) - MC    0.517 1.32 33  0.392  0.6977
```

We follow the point estimates with confidence intervals for the pairwise differences:

```
confint(pairwise.model)
#  contrast     estimate   SE df lower.CL upper.CL
#  DC - (DC+MK)   15.358 1.32 33    12.68     18.0
#  DC - MC        15.875 1.32 33    13.19     18.6
#  (DC+MK) - MC    0.517 1.32 33    -2.17      3.2
#
# Confidence level used: 0.95
```

This is strong evidence that `DC` produces a larger effect than `DC+MK` and `MC`. Finally, let's produce the diagnostic plots to ensure our model is adequate.

```
op <- par(mfrow=c(1,2))
qqnorm(resid(means.model), main='Normal Probability Plot')
qqline(resid(means.model), col='orange')
plot(x=predict(means.model), y=resid(means.model), main='Fitted vs.
↪  Residual')
```




```
par(op)
```

Both diagnostic plots fail to show any strong evidence of deviation from our modelling assumptions. This provides us with enough information to draw the same conclusion as the text — dark chocolate "DC" results in a higher mean blood antioxidant capacity than the other two chocolates.

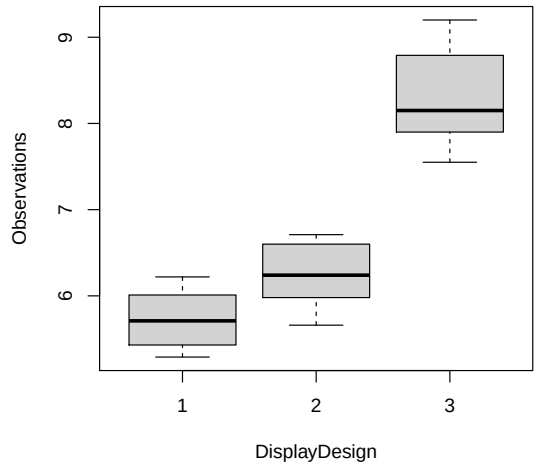
End-Aisle Experiment of Section 3.8.2

The end-aisle experiment of Section 3.8.2 compares three different display designs in terms of the percent increase in sales. Here, the design is encoded using the numbers 1, 2, and 3 so we must be careful that those are recorded as factors:

```
str(Table3.14)
# 'data.frame': 15 obs. of  2 variables:
#  $ DisplayDesign: chr  "1" "2" "3" "1" ...
#  $ Observations : num  5.43 6.24 8.79 5.71 6.71 9.2 6.22 5.98 7.9
↪  6.01 ...
```

A visual inspection of the distributions of sales-increases suggests that `DisplayDesign` has an effect:

```r
boxplot(Observations ~ DisplayDesign, data=Table3.14)
```



Let's start by running the ANOVA general F-test:

```r
model <- aov(Observations ~ DisplayDesign, data=Table3.14)
summary(model)
#               Df Sum Sq Mean Sq F value   Pr(>F)
# DisplayDesign  2  18.78   9.392   35.77 8.78e-06 ***
# Residuals     12   3.15   0.263
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We are able to reject the null hypothesis that all for the treatments have equal means. Next, we estimate the mean for each `DisplayDesign`. We can use the regular `aov` function if we switch to the cell means model:

```r
means.model <- lm(Observations ~ DisplayDesign - 1, data=Table3.14)
summary(means.model)
#
# Call:
# lm(formula = Observations ~ DisplayDesign - 1, data = Table3.14)
#
# Residuals:
#    Min     1Q Median     3Q    Max
# -0.768 -0.360 -0.022  0.417  0.882
#
# Coefficients:
#                Estimate Std. Error t value Pr(>|t|)
# DisplayDesign1   5.7320     0.2291   25.02 1.01e-11 ***
# DisplayDesign2   6.2380     0.2291   27.22 3.72e-12 ***
```

```
# DisplayDesign3   8.3180     0.2291   36.30 1.22e-13 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 0.5124 on 12 degrees of freedom
# Multiple R-squared:  0.9955,  Adjusted R-squared:  0.9944
# F-statistic: 894.8 on 3 and 12 DF,  p-value: 2.273e-14
```

Then, we can construct the confidence intervals for the means with `confint`:

```
confint(means.model)
#                   2.5 %   97.5 %
# DisplayDesign1 5.232736 6.231264
# DisplayDesign2 5.738736 6.737264
# DisplayDesign3 7.818736 8.817264
```

Alternatively, we could use the `emmeans` package to do this from the effects model. We indicate that we want the means with respect to `DisplayDesign`:

```
ls.model <- emmeans(model, ~DisplayDesign)
print(ls.model)
#  DisplayDesign emmean    SE df lower.CL upper.CL
#  1               5.73 0.229 12     5.23     6.23
#  2               6.24 0.229 12     5.74     6.74
#  3               8.32 0.229 12     7.82     8.82
#
# Confidence level used: 0.95
```

Next, we construct the confidence intervals:

```
confint(ls.model)
#  DisplayDesign emmean    SE df lower.CL upper.CL
#  1               5.73 0.229 12     5.23     6.23
#  2               6.24 0.229 12     5.74     6.74
#  3               8.32 0.229 12     7.82     8.82
#
# Confidence level used: 0.95
```

Both methods are presented because the cell means model is often easier but the `emmeans` package will be of more use as we go forward. Note that, in constructing 95% confidence intervals, it appears that design 3's confidence interval is disjoint from those from designs 1 and 2. Looking at all pairwise comparisons, it appears that design 3 produces the largest mean increase in sales:

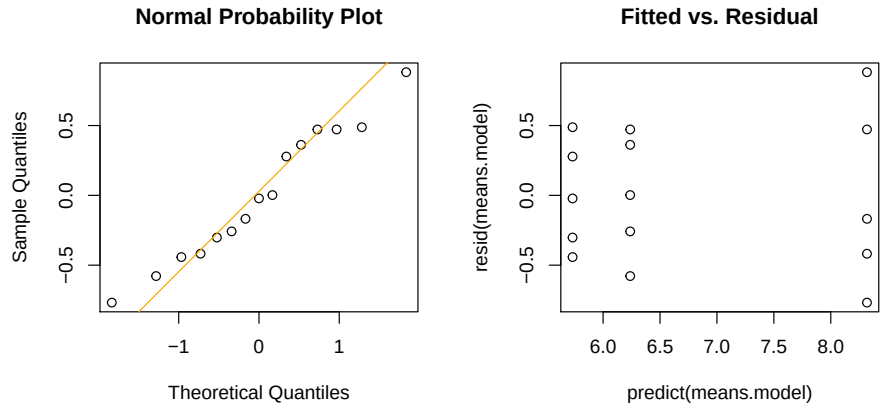
```
pairwise.model <- pairs(ls.model, adjust='none')
print(pairwise.model)
#  contrast estimate    SE df t.ratio p.value
#  1 - 2      -0.506 0.324 12 -1.561  0.1444
#  1 - 3      -2.586 0.324 12 -7.980  <.0001
#  2 - 3      -2.080 0.324 12 -6.419  <.0001
```

We can also construct confidence intervals on the pairwise differences.

```
confint(pairwise.model)
#  contrast estimate    SE df lower.CL upper.CL
#  1 - 2      -0.506 0.324 12    -1.21     0.20
#  1 - 3      -2.586 0.324 12    -3.29    -1.88
#  2 - 3      -2.080 0.324 12    -2.79    -1.37
#
# Confidence level used: 0.95
```

Next, let's examine the diagnostic plots:

```
op <- par(mfrow=c(1,2))
qqnorm(resid(means.model), main='Normal Probability Plot')
qqline(resid(means.model), col='orange')
plot(x=predict(means.model), y=resid(means.model), main='Fitted vs.
↪  Residual')
```




```
par(op)
```

There may be a slight increase in variance displayed by the fitted versus residual plots, but no strong evidence of model misspecification.

Smelting Experiment of Section 3.8.3

The aluminum smelting experiment of Section 3.8.3 describes four different ratio control algorithms governing the introduction of alumina to a cell where a response related to cell voltage was recorded. Here, we're going to model the standard deviation of cell voltage to determine which algorithm is most consistent. First, let's check that `RatioControlAlgorithm` is a factor:

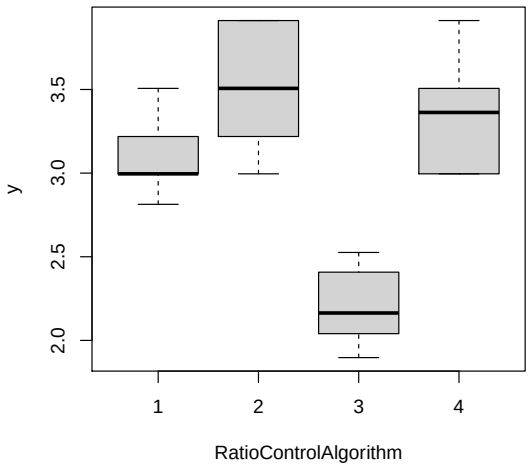
```
str(Table3.16)
# 'data.frame': 24 obs. of  3 variables:
#  $ RatioControlAlgorithm: chr  "1" "1" "1" "1" ...
#  $ AvgVoltage           : num  4.93 4.86 4.75 4.95 4.79 4.88 4.85
↪  4.91 4.79 4.85 ...
#  $ StDevVoltage         : num  0.05 0.04 0.05 0.06 0.03 0.05 0.04
↪  0.02 0.03 0.05 ...
```

As suggested by the text, we apply the negative log transform to the `StDevVoltage` before fitting our model:

```
y <- -log(Table3.16$StDevVoltage)
summary(aov(y ~ RatioControlAlgorithm, data=Table3.16))
#                       Df Sum Sq Mean Sq F value   Pr(>F)
# RatioControlAlgorithm  3  6.166  2.0554   21.96 1.54e-06 ***
# Residuals             20  1.872  0.0936
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

A quick visual inspection using box plots strongly suggests that algorithm 3 produces the largest noise, whereas the others all appear about the same:

```
boxplot(y ~ RatioControlAlgorithm, data=Table3.16)
```



Random Effects Model of Section 3.9 (Example 3.10)

For the loom experiment of Section 3.9 we wish to model the looms as a population, and hence wish to use a random effects model. We can get the ANOVA values for balanced designs using the `Error` term and the normal `aov` function:

```
model <- aov(Strength ~ Error(Looms), data=Example3.10)
summary(model)
#
# Error: Looms
#           Df Sum Sq Mean Sq F value Pr(>F)
# Residuals  3  89.19   29.73
#
# Error: Within
#           Df Sum Sq Mean Sq F value Pr(>F)
# Residuals 12  22.75   1.896
```

In this method we would have to manually construct the mean square errors, F-statistic, and p-value. The other method for fitting this model is using the `lme4` package (Bates et al. 2019) and `lmerTest` package (Kuznetsova, Brockhoff, and Christensen 2019) for fitting random or mixed effects models. In `lme4`, to indicate a random intercept for each level of loom we use `(1|Looms)` in the formula as follows:

```
library(lme4)
library(lmerTest)

model <- lmer(
    Strength ~ (1|Looms),
    data=Example3.10,
    REML=TRUE
)
print(summary(model))
# Linear mixed model fit by REML. t-tests use Satterthwaite's method [
# lmerModLmerTest]
# Formula: Strength ~ (1 | Looms)
#    Data: Example3.10
#
# REML criterion at convergence: 63.2
#
# Scaled residuals:
#      Min       1Q   Median       3Q      Max
# -1.38018 -0.57260 -0.04342  0.82574  1.52491
#
# Random effects:
#  Groups   Name        Variance Std.Dev.
#  Looms    (Intercept) 6.958    2.638
#  Residual             1.896    1.377
# Number of obs: 16, groups:  Looms, 4
#
# Fixed effects:
#             Estimate Std. Error     df t value Pr(>|t|)
# (Intercept)   95.438      1.363  3.000   70.01 6.42e-06 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
print(rand(model))
# ANOVA-like table for random-effects: Single term deletions
#
# Model:
# Strength ~ (1 | Looms)
#             npar  logLik    AIC    LRT Df Pr(>Chisq)
# <none>         3 -31.597 69.193
# (1 | Looms)    2 -37.745 79.489 12.296  1  0.0004539 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This matches the JMP output of Table 3.20. We can generate confidence intervals for the variance components easily:

```
confint(model, oldNames=FALSE)
#                            2.5 %    97.5 %
# sd_(Intercept)|Looms  1.1293759  5.748927
# sigma                 0.9672533  2.184005
# (Intercept)          92.4393299 98.435668
```

Above, the confidence intervals for the standard deviations differ from JMP in that these are based on the profile likelihood.

Kruskal-Wallis Test of Section 3.11

Section 3.11 describes the non-parametric Kruskal-Wallis test. This test doesn't assume normality and is very easy to use in R. Here, we run the test against the plasma etch experiment data of Table 3.1:

```
kruskal.test(Observation ~ Power, data=Table3.1)
#
#   Kruskal-Wallis rank sum test
#
# data:  Observation by Power
# Kruskal-Wallis chi-squared = 16.907, df = 3, p-value = 0.0007386
```

We find significant difference between the treatments.

References

Bates, Douglas, Marint Maechler, Ben Bolker, Steven Walker, Christensen Rune Jaubo Bojesen, Henrik Singmann, Bin Dai, et al. 2019. *Lme4: Linear Mixed-Effects Models Using 'Eigen' and S4*. https://cran.r-project.org/package=lme4.

Kuznetsova, Alexandra, Per Bruun Brockhoff, and Rune Jaubo Bojesen Christensen. 2019. *LmerTest: Tests in Linear Mixed Effects Models*. https://cran.r-project.org/package=lmerTest.

Lenth, Russell, Henrik Singmann, Jonathon Love, Paul Buerkner, and Maxime Herve. 2019. *Emmeans: Estimated Marginal Means, Aka Least-Squares Means*. https://cran.r-project.org/package=emmeans.

Chapter 4

Randomized Blocks, Latin Squares, and Related Designs

In Chapter 4 the text introduces designs that can accomodate investigations where complete randomization isn't feasible. These designs are useful when you want to estimate the effect of a factor and avoid bias due to other nuissance factors.

Example 4.1

Example 4.1 describes an experiment where batches of resin are extruded at different pressures and the proportion of defective grafts are recorded. The goal of the experiment is to maximize yield by minimizing the number of defects (`Flicks`). The structure of the data is presented below:

```
str(Table4.3)
# 'data.frame': 24 obs. of  3 variables:
#  $ ExtrusionPressure: chr  "8500" "8700" "8900" "9100" ...
#  $ BatchOfResin     : chr  "1" "1" "1" "1" ...
#  $ Flicks           : num  90.3 92.5 85.5 82.5 89.2 89.5 90.8 89.5
↪  98.2 90.6 ...
```

In generating the ANOVA table we can either include `BatchOfResin` and ignore the p-value or we can mark it with `Error` to indicate each level is a block:

```
model <- aov(Flicks ~ Error(BatchOfResin) + ExtrusionPressure,
↪  data=Table4.3)
print(summary(model))
#
```

```
# Error: BatchOfResin
#           Df Sum Sq Mean Sq F value Pr(>F)
# Residuals  5  192.2   38.45
#
# Error: Within
#                   Df Sum Sq Mean Sq F value  Pr(>F)
# ExtrusionPressure  3  178.2   59.39   8.107 0.00192 **
# Residuals         15  109.9    7.33
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

What if we hadn't recognized that this was a completely randomized block design? That would yield the following ANOVA table:

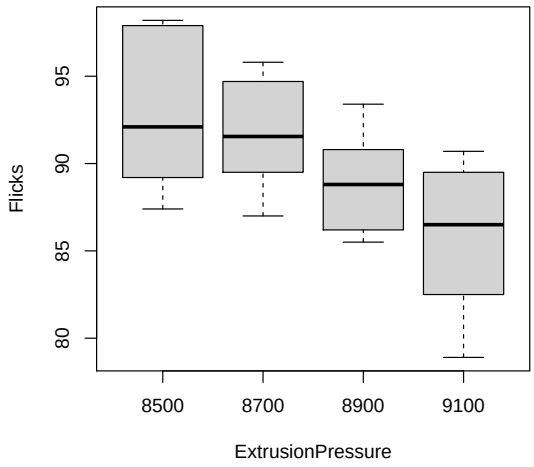
```
summary(aov(Flicks ~ ExtrusionPressure, data=Table4.3))
#                   Df Sum Sq Mean Sq F value Pr(>F)
# ExtrusionPressure  3  178.2   59.39   3.931 0.0234 *
# Residuals         20  302.1   15.11
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

To estimate the means we turn to the `emmeans` package (Lenth et al. 2019):

```
library(emmeans)
model <- aov(Flicks ~ ExtrusionPressure + BatchOfResin, data=Table4.3)
exmodel <- emmeans(model, ~ ExtrusionPressure)
print(exmodel)
#  ExtrusionPressure emmean  SE df lower.CL upper.CL
#  8500                92.8 1.1 15     90.5     95.2
#  8700                91.7 1.1 15     89.3     94.0
#  8900                88.9 1.1 15     86.6     91.3
#  9100                85.8 1.1 15     83.4     88.1
#
# Results are averaged over the levels of: BatchOfResin
# Confidence level used: 0.95
blmodel <- emmeans(model, ~ BatchOfResin)
print(blmodel)
#  BatchOfResin emmean   SE df lower.CL upper.CL
#  1             87.7 1.35 15     84.8     90.6
#  2             89.8 1.35 15     86.9     92.6
#  3             91.0 1.35 15     88.1     93.9
#  4             90.5 1.35 15     87.7     93.4
#  5             85.3 1.35 15     82.4     88.2
#  6             94.5 1.35 15     91.6     97.3
#
# Results are averaged over the levels of: ExtrusionPressure
# Confidence level used: 0.95
```

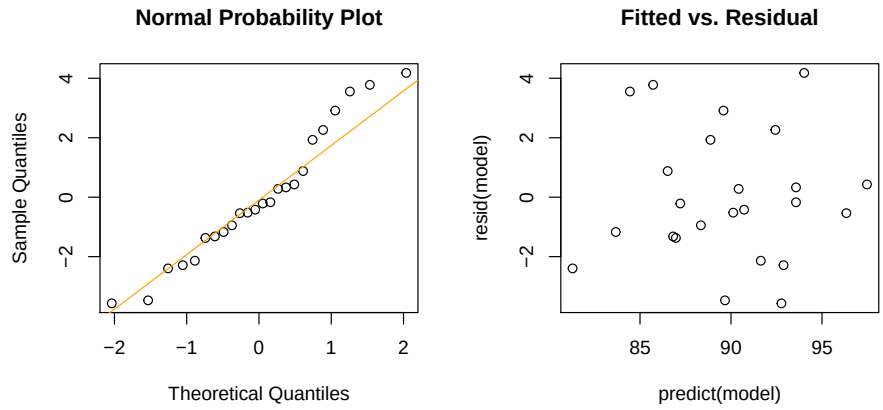
We follow this with a boxplot for the defects:

```
boxplot(Flicks ~ ExtrusionPressure, data=Table4.3)
```



Consider the diagnostic plots for Example 4.1:

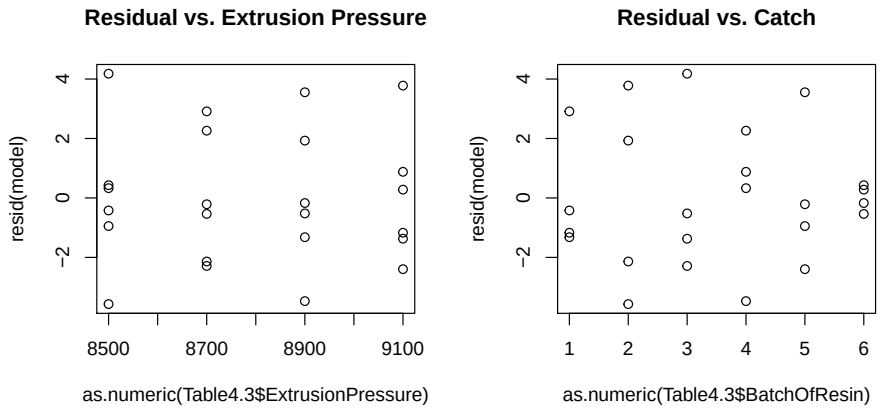
```
op <- par(mfrow=c(1,2))
qqnorm(resid(model), main='Normal Probability Plot')
qqline(resid(model), col='orange')
plot(x=predict(model), y=resid(model), main='Fitted vs. Residual')
```




```
par(op)
```

These plots do not show strong signs of any problem. The plot of residual versus each of the factors would be informative if some model misspecificaiton had occurred, but as in Figure 4.6, we find no strong evidence of problems from such a plot:

```
op <- par(mfrow=c(1,2))
plot(y=resid(model), x=as.numeric(Table4.3$ExtrusionPressure),
↪  main='Residual vs. Extrusion Pressure')
plot(y=resid(model), x=as.numeric(Table4.3$BatchOfResin),
↪  main='Residual vs. Catch')
```




```
par(op)
```

Let's assume the blocks are random effects, then we arrive at the following model:

```
library(lme4)
library(lmerTest)
model <- lmer(
    Flicks ~ -1 + ExtrusionPressure + (1|BatchOfResin),
    data=Table4.3,
    REML=TRUE
)
print(summary(model))
# Linear mixed model fit by REML. t-tests use Satterthwaite's method [
# lmerModLmerTest]
# Formula: Flicks ~ -1 + ExtrusionPressure + (1 | BatchOfResin)
#    Data: Table4.3
#
# REML criterion at convergence: 112
#
# Scaled residuals:
#      Min       1Q   Median       3Q      Max
# -1.32253 -0.64269  0.01068  0.54202  1.62882
#
# Random effects:
#  Groups       Name        Variance Std.Dev.
#  BatchOfResin (Intercept) 7.781    2.789
#  Residual                 7.326    2.707
```

```
# Number of obs: 24, groups:  BatchOfResin, 6
#
# Fixed effects:
#                       Estimate Std. Error     df t value Pr(>|t|)
↪
# ExtrusionPressure8500   92.817      1.587 11.136   58.49 3.24e-15
↪  ***
# ExtrusionPressure8700   91.683      1.587 11.136   57.78 3.71e-15
↪  ***
# ExtrusionPressure8900   88.917      1.587 11.136   56.04 5.22e-15
↪  ***
# ExtrusionPressure9100   85.767      1.587 11.136   54.05 7.79e-15
↪  ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Correlation of Fixed Effects:
#             EP8500 EP8700 EP8900
# ExtrsnP8700 0.515
# ExtrsnP8900 0.515  0.515
# ExtrsnP9100 0.515  0.515  0.515
print(rand(model))
# ANOVA-like table for random-effects: Single term deletions
#
# Model:
# Flicks ~ ExtrusionPressure + (1 | BatchOfResin) - 1
#                    npar  logLik    AIC    LRT Df Pr(>Chisq)
# <none>                6 -56.021 124.04
# (1 | BatchOfResin)    5 -59.114 128.23 6.1853  1    0.01288 *
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Confidence intervals on the variance components and the fixed effects can also be evaluated:

```
confint(model, oldNames=FALSE)
#                                 2.5 %    97.5 %
# sd_(Intercept)|BatchOfResin  1.136322  5.540193
# sigma                        1.840242  3.564370
# ExtrusionPressure8500       89.726101 95.907233
# ExtrusionPressure8700       88.592767 94.773900
# ExtrusionPressure8900       85.826101 92.007233
# ExtrusionPressure9100       82.676101 88.857233
```

Example 4.2

In Example 4.2 an experimenter is studying the effects five different formulation of rocket propellant on the burn rate of that propellant (see Table 4.9 in the textbook). Formulations are constructed from batches that can only supply five formulations in all. Also, the formulations are prepared by different operators. The two levels of blocking — the batch and the operator — require the use of a Latin square design. First, let's look at the structure of the data:

```
str(Table4.9)
# 'data.frame': 25 obs. of  4 variables:
#  $ Batch      : chr  "1" "2" "3" "4" ...
#  $ Operator   : chr  "1" "1" "1" "1" ...
#  $ Formulation: chr  "A" "B" "C" "D" ...
#  $ BurnRate   : num  24 17 18 26 22 20 24 38 31 30 ...
```

Next, let's conduct the ANOVA test to see if `Formulation` has an effect. Again, wrapping `Batch` and `Operator` in `Error` is optional and just prevents rendering p-values for them.

```
y <- Table4.9$BurnRate - 25
model <- aov(y ~ Error(Batch + Operator) + Formulation, data=Table4.9)
summary(model)
#
# Error: Batch
#           Df Sum Sq Mean Sq F value Pr(>F)
# Residuals  4     68      17
#
# Error: Operator
#           Df Sum Sq Mean Sq F value Pr(>F)
# Residuals  4    150    37.5
#
# Error: Within
#             Df Sum Sq Mean Sq F value  Pr(>F)
# Formulation  4    330   82.50   7.734 0.00254 **
# Residuals   12    128   10.67
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In this situation the design was provided, but you can use R to find Latin squares and Graeco-Latin squares. The package `agricolae` (Mendiburu 2019) contains functions for finding such designs. Constructing a Latin square can be done with `design.lsd`:

```r
library(agricolae)

trt <- c("A", "B", "C", "D")
outdesign <- design.lsd(trt)
print(outdesign$sketch)
#      [,1] [,2] [,3] [,4]
# [1,] "A"  "D"  "B"  "C"
# [2,] "B"  "A"  "C"  "D"
# [3,] "C"  "B"  "D"  "A"
# [4,] "D"  "C"  "A"  "B"
```

The `agricolae` package can also be used to construct Graeco-Latin squares with `design.graeco` and a host of other designs (balanced incomplete block designs, split plots, strip plots, etc.).

Example 4.3

In Example 4.3 the rocket propellant data is analyzed with an additional blocking factor: `TestAssembly`. The ANOVA table for this model is presented below:

```r
y <- Table4.20$BurnRate - 25
model <- aov(y ~ Error(Batch + Operator + TestAssembly) + Formulation,
↪  data=Table4.20)
summary(model)
#
# Error: Batch
#           Df Sum Sq Mean Sq F value Pr(>F)
# Residuals  4     68      17
#
# Error: Operator
#           Df Sum Sq Mean Sq F value Pr(>F)
# Residuals  4    150    37.5
#
# Error: TestAssembly
#           Df Sum Sq Mean Sq F value Pr(>F)
# Residuals  4     62    15.5
#
# Error: Within
#             Df Sum Sq Mean Sq F value  Pr(>F)
# Formulation  4    330   82.50      10 0.00334 **
# Residuals    8     66    8.25
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Section 4.4 — Balanced Incomplete Block Design

The data in Table 4.22 in the textbook is from an experiment when an engineer tests four types of catalyst to optimize reaction time. Batches of material are only enough for three runs, so we must treat the batches as blocks. This is a balanced incomplete block design and is not orthogonal. Because of this, `ReactionTime ~ Batch + Catalyst` and `ReactionTime ~ Catalyst + Batch` give different results. Taking `Batch` before `Catalyst` adjusts `Catalyst` for the effect of `Batch`. This is what we want:

```
model <- aov(ReactionTime ~ Batch + Catalyst, data=Table4.22)
summary(model)
#             Df Sum Sq Mean Sq F value  Pr(>F)
# Batch        3  55.00  18.333   28.20 0.00147 **
# Catalyst     3  22.75   7.583   11.67 0.01074 *
# Residuals    5   3.25   0.650
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Compare this to the other order and we get the unadjusted treatment and adjusted block in Table 4.25:

```
model <- aov(ReactionTime ~ Catalyst + Batch, data=Table4.22)
summary(model)
#             Df Sum Sq Mean Sq F value   Pr(>F)
# Catalyst     3  11.67   3.889   5.983 0.041463 *
# Batch        3  66.08  22.028  33.889 0.000953 ***
# Residuals    5   3.25   0.650
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Interestingly, if you wrap `Batch` with `Error` then R evaluates it first and the order does not matter.

Let's fit a model for parameter estimation. I'm going to use `emmeans` so we can't use the `Error` function:

```
library(emmeans)
model <- aov(ReactionTime ~ Batch + Catalyst, Table4.22)
print(summary(model))
#             Df Sum Sq Mean Sq F value  Pr(>F)
```

```
# Batch        3  55.00  18.333   28.20 0.00147 **
# Catalyst     3  22.75   7.583   11.67 0.01074 *
# Residuals    5   3.25   0.650
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
camodel <- emmeans(model, ~Catalyst)
print(camodel)
#  Catalyst emmean    SE df lower.CL upper.CL
#  1          71.4 0.487  5     70.1     72.6
#  2          71.6 0.487  5     70.4     72.9
#  3          72.0 0.487  5     70.7     73.3
#  4          75.0 0.487  5     73.7     76.3
#
# Results are averaged over the levels of: Batch
# Confidence level used: 0.95
bamodel <- emmeans(model, ~Batch)
print(bamodel)
#  Batch emmean    SE df lower.CL upper.CL
#  1       73.4 0.487  5     72.1     74.6
#  2       75.5 0.487  5     74.2     76.8
#  3       68.6 0.487  5     67.4     69.9
#  4       72.5 0.487  5     71.2     73.8
#
# Results are averaged over the levels of: Catalyst
# Confidence level used: 0.95
print(pairs(camodel, adjust='tukey'))
#  contrast estimate    SE df t.ratio p.value
#  1 - 2      -0.250 0.698  5 -0.358  0.9825
#  1 - 3      -0.625 0.698  5 -0.895  0.8085
#  1 - 4      -3.625 0.698  5 -5.192  0.0130
#  2 - 3      -0.375 0.698  5 -0.537  0.9462
#  2 - 4      -3.375 0.698  5 -4.834  0.0175
#  3 - 4      -3.000 0.698  5 -4.297  0.0281
#
# Results are averaged over the levels of: Batch
# P value adjustment: tukey method for comparing a family of 4
↪  estimates
```

References

Lenth, Russell, Henrik Singmann, Jonathon Love, Paul Buerkner, and Maxime Herve. 2019. *Emmeans: Estimated Marginal Means, Aka Least-Squares Means*. https://cran.r-project.org/package=emmeans.

Mendiburu, Felipe de. 2019. *Agricolae: Statistical Procedures for Agricultural Research*. https://cran.r-project.org/package=agricolae.

Chapter 5

Introduction to Factorial Designs

Chapter 5 in the text introduces factorial designs. Factorial designs allow us to estimate interactions between factors and are efficient configurations for running experiments. Here, we will focus on construction and analysis of the designs presented in Chapter 5 of the text.

Example 5.1

Example 5.1 examines how the effective life of batteries is affected by the material type (three levels) and the temperature (three levels). First, let's check that the structure of the data in `Table5.1` is appropriate:

```
str(Table5.1)
# 'data.frame': 36 obs. of  3 variables:
#  $ MaterialType: chr  "1" "1" "2" "2" ...
#  $ Temperature : num  15 15 15 15 15 15 15 15 15 15 ...
#  $ BatteryLife : num  130 74 150 159 138 168 155 180 188 126 ...
```

Here, `Temperature` is numeric and not a factor, so we need to convert it before running the model:

```
df <- Table5.1
df$Temperature <- factor(df$Temperature)
```

Next, we use a partial F-test to check if the full model is better than the null model (having only an intercept term) using the `anova` function:

```
null.model <- aov(BatteryLife~1, data=df)
full.model <- aov(BatteryLife ~ MaterialType + Temperature +
↪  MaterialType:Temperature, data=df)
anova(null.model, full.model)
# Analysis of Variance Table
#
# Model 1: BatteryLife ~ 1
# Model 2: BatteryLife ~ MaterialType + Temperature +
↪  MaterialType:Temperature
#   Res.Df   RSS Df Sum of Sq      F    Pr(>F)
# 1     35 77647
# 2     27 18231  8     59416 10.999 9.426e-07 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

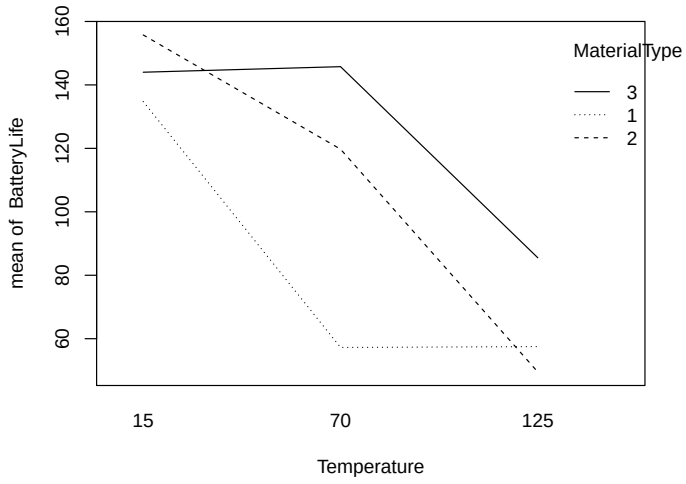
We can compare any two models using the `anova` function in this way. The ANOVA table for the full model is presented next:

```
summary(full.model)
#                          Df Sum Sq Mean Sq F value   Pr(>F)
# MaterialType              2  10684    5342   7.911  0.00198 **
# Temperature               2  39119   19559  28.968 1.91e-07 ***
# MaterialType:Temperature  4   9614    2403   3.560  0.01861 *
# Residuals                27  18231     675
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here we see the interaction is significant even after controlling for `MaterialType` and `Temperature`.

What does this interaction look like? We can plot it in two ways. The easier way is to use the built-in `interaction.plot` function:

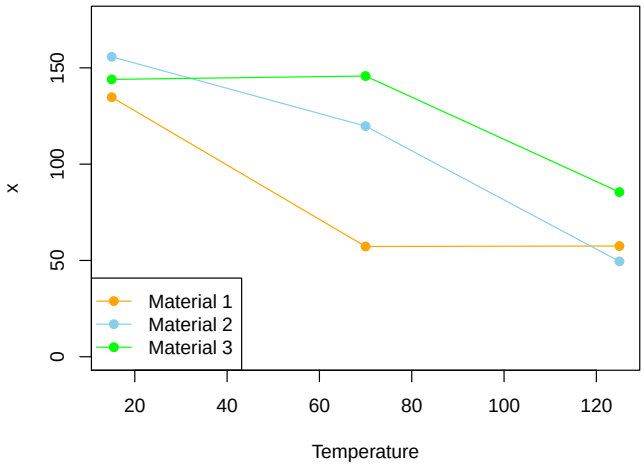
```
with(df, {
    interaction.plot(Temperature, MaterialType, BatteryLife)
})
```



With more work we can make a more colorful plot. For example:

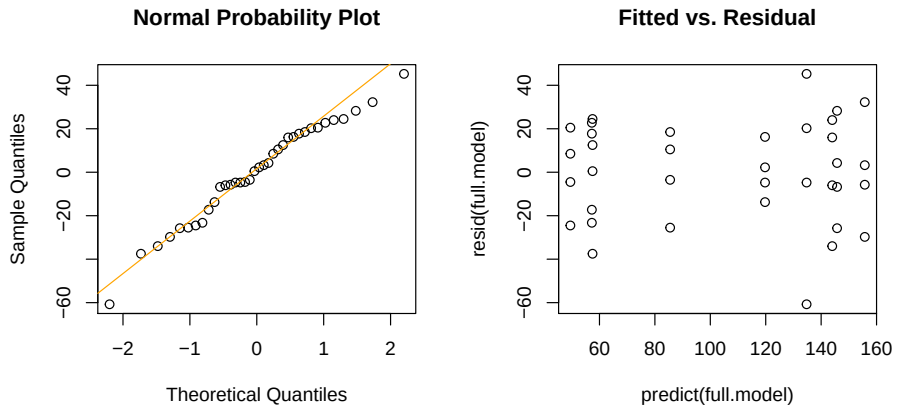
```
af <- aggregate(
    df$BatteryLife,
    by=list(
        'Temperature'=Table5.1$Temperature,
        'MaterialType'=df$MaterialType
    ),
    FUN=mean
)
plot(
    af[af[, 'MaterialType'] == 1, c('Temperature', 'x')],
    type='o',
    ylim=c(0,175),
    col='orange',
    pch=19
)
lines(
    af[af[, 'MaterialType'] == 2, c('Temperature', 'x')],
    type='o',
    col='skyblue',
    pch=19
)
lines(
    af[af[, 'MaterialType'] == 3, c('Temperature', 'x')],
    type='o',
    col='green',
    pch=19
)
legend(
    'bottomleft',
    legend=c('Material 1', 'Material 2', 'Material 3'),
    col=c('orange', 'skyblue', 'green'),
    lty=rep(1,3),
```

```
    pch=rep(19,3)
)
```



Next, let's check the model assumptions with diagnostic plots. The normal probability plot and the fitted versus residual do not show any strong signs of trouble:

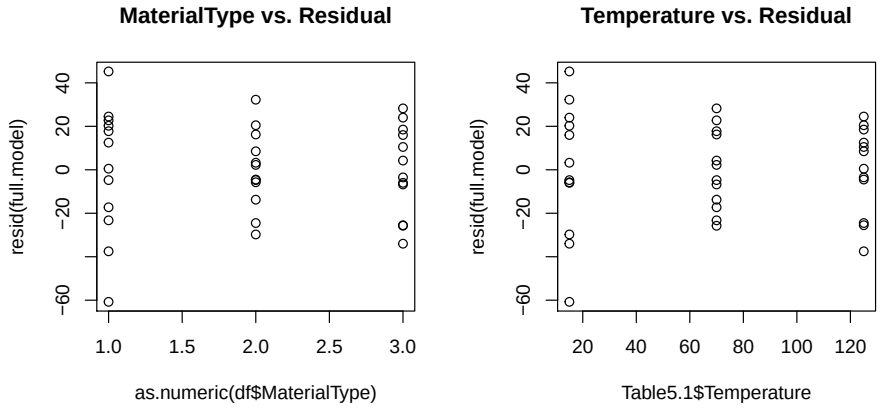
```
op <- par(mfrow=c(1,2))
qqnorm(resid(full.model), main='Normal Probability Plot')
qqline(resid(full.model), col='orange')
plot(x=predict(full.model), y=resid(full.model), main='Fitted vs.
↪  Residual')
```



```
par(op)
```

Next, let's look at the residuals by each factor to check for model misspecification:

```
op <- par(mfrow=c(1,2))
plot(x=as.numeric(df$MaterialType), y=resid(full.model),
↪  main='MaterialType vs. Residual')
plot(x=Table5.1$Temperature, y=resid(full.model), main='Temperature
↪  vs. Residual')
```




```
par(op)
```

These plots also do not look too terribly troubling.

Example 5.2

Table 5.10 in the textbook gives the data for an experiment where the impurity present in a chemical product is measured as the pressure and temperature of the process are varied. We use Tukey's non-additivity test to model this as

$$y_{ij} = \mu + \tau_i + \beta_j + \gamma\tau_i\beta_j \quad \begin{cases} i = 1, 2, \ldots, a \\ j = 1, 2, \ldots, b \end{cases}$$

To do this we make use of the `tukey.1df` function from the `dae` package (Brien 2018):

```
library(dae)

df <- Table5.10
df[,'fTemperature'] <- factor(df$Temperature)
df[,'fPressure'] <- factor(df$Pressure)
model <- aov(Impurity ~ fTemperature + fPressure, data=df)
tukey.1df(model, data=df)
# $Tukey.SS
# [1] 0.09852217
#
# $Tukey.F
# [1] 0.3626943
#
# $Tukey.p
# [1] 0.5660026
#
# $Devn.SS
# [1] 1.901478
```

Compare this to the output in Table 5.11.

Example 5.3

Table 5.13 contains fill height deviation data for a soft drink bottler. This variation is potentially driven by the percent carbonation, the operating pressure of the filler, and the speed of the line. Note that `Table5.13` is all numeric data so we must first convert it to factors:

```
str(Table5.13)
# 'data.frame': 24 obs. of  4 variables:
#  $ PctCarbonation     : num  10 10 12 12 14 14 10 10 12 12 ...
#  $ OperatingPressure  : num  25 25 25 25 25 25 25 25 25 25 ...
#  $ LineSpeed          : num  200 200 200 200 200 200 250 250 250 250
↪  ...
#  $ FillHeightDeviation: num  -3 -1 0 1 5 4 -1 0 2 1 ...
```

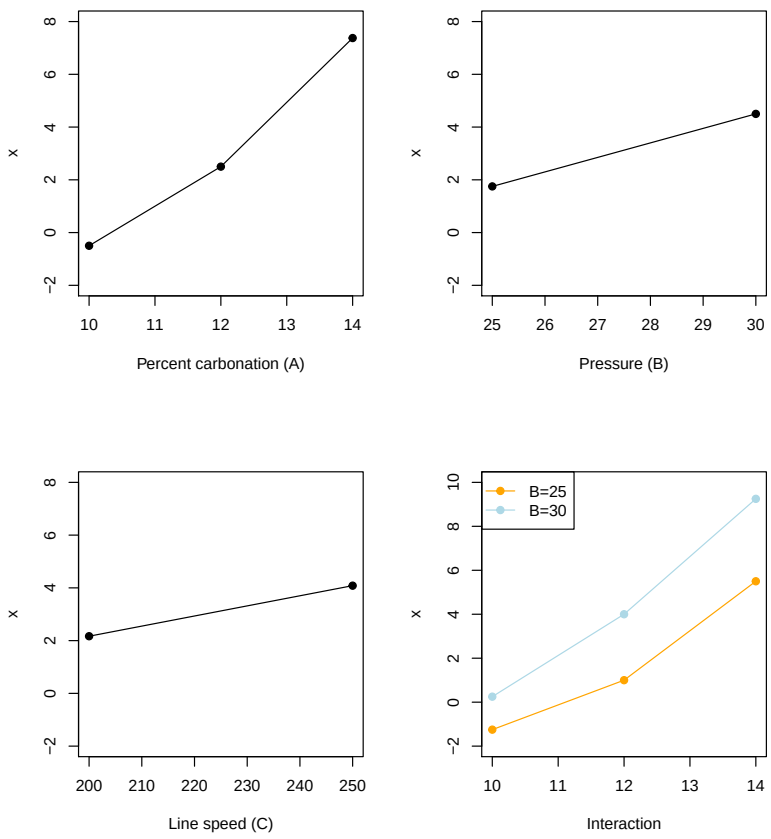
Generating the factors, we arrive at the following model:

```
with(Table5.13, {
    A <- factor(PctCarbonation)
    B <- factor(OperatingPressure)
    C <- factor(LineSpeed)
    model <- aov(FillHeightDeviation ~ A*B*C)
    print(summary(model))
```

```
})
#              Df Sum Sq Mean Sq F value   Pr(>F)
# A             2 252.75  126.38 178.412 1.19e-09 ***
# B             1  45.37   45.37  64.059 3.74e-06 ***
# C             1  22.04   22.04  31.118  0.00012 ***
# A:B           2   5.25    2.62   3.706  0.05581 .
# A:C           2   0.58    0.29   0.412  0.67149
# B:C           1   1.04    1.04   1.471  0.24859
# A:B:C         2   1.08    0.54   0.765  0.48687
# Residuals    12   8.50    0.71
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Next, let's construct Figure 5.16:

```
op <- par(mfrow=c(2,2))
attach(Table5.13)
levels <- list(
    'Percent carbonation (A)'=PctCarbonation,
    'Pressure (B)'=OperatingPressure,
    'Line speed (C)'=LineSpeed
)
for(s in names(levels)){
    df <- aggregate(
        FillHeightDeviation,
        by=list('A'=levels[[s]]),
        FUN=mean
    )
    plot(df, type='o', pch=19, xlab=s, ylim=c(-2,8))
}
df <- aggregate(
    FillHeightDeviation,
    by=list('A'=PctCarbonation, 'B'=OperatingPressure),
    FUN=mean
)
detach(Table5.13)

plot(df[df[,'B'] == 25,c('A','x')], type='o', pch=19, col='orange',
↪  ylim=c(-2, 10), xlab='Interaction')
lines(df[df[,'B'] == 30,c('A','x')], type='o', pch=19,
↪  col='lightblue')
legend('topleft', legend=c('B=25','B=30'), lty=c(1,1), pch=c(19, 19),
↪  col=c('orange', 'lightblue'))
```




```
par(op)
```

Example 5.4

We return again the the battery life experiment of Example 5.1, and analyze the effect of treating temperature as a continuous factor with quadratic effects. Note that we'll code the `Temperature` factor to be in $[-1, 1]$.

```
df <- Table5.1
df$Temperature <- (df$Temperature - 70) / 55
df$MaterialType <- factor(df$MaterialType)
contrasts(df$MaterialType) <- as.matrix(cbind(c(1,0,-1), c(0,1,-1)))
model <- lm(BatteryLife ~ Temperature + MaterialType +
↪  I(Temperature^2) + Temperature:MaterialType +
↪  I(Temperature^2):MaterialType, data=df)
anova(model)
```

```
# Analysis of Variance Table
#
# Response: BatteryLife
#                              Df Sum Sq Mean Sq F value    Pr(>F)
↪
# Temperature                   1  39043   39043 57.8227 3.525e-08
↪  ***
# MaterialType                  2  10684    5342  7.9114  0.001976 **
↪
# I(Temperature^2)              1     76      76  0.1126  0.739753
↪
# Temperature:MaterialType      2   2315    1158  1.7143  0.199109
↪
# MaterialType:I(Temperature^2)  2   7299    3649  5.4047  0.010612 *
↪
# Residuals                    27  18231     675
↪
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

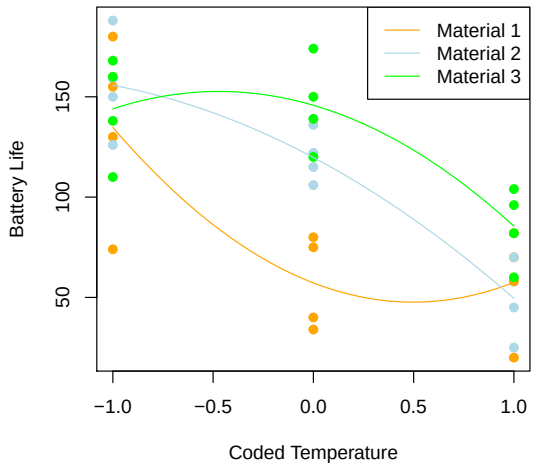
Next, let's examine the coefficients of the regression model:

```
summary(model)
#
# Call:
# lm(formula = BatteryLife ~ Temperature + MaterialType +
↪  I(Temperature^2) +
#     Temperature:MaterialType + I(Temperature^2):MaterialType,
#     data = df)
#
# Residuals:
#     Min      1Q  Median      3Q     Max
# -60.750 -14.625   1.375  17.937  45.250
#
# Coefficients:
#                               Estimate Std. Error t value Pr(>|t|)
↪
# (Intercept)                    107.583      7.501  14.342 3.79e-14
↪  ***
# Temperature                    -40.333      5.304  -7.604 3.53e-08
↪  ***
# MaterialType1                  -50.333     10.608  -4.745 6.05e-05
↪  ***
# MaterialType2                   12.167     10.608   1.147  0.26148
↪
# I(Temperature^2)                -3.083      9.187  -0.336  0.73975
↪
# Temperature:MaterialType1        1.708      7.501   0.228  0.82156
↪
# Temperature:MaterialType2      -12.792      7.501  -1.705  0.09962
↪  .
```

```
# MaterialType1:I(Temperature^2)   41.958     12.992   3.229  0.00325
↪  **
# MaterialType2:I(Temperature^2)  -14.042     12.992  -1.081  0.28936
↪
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 25.98 on 27 degrees of freedom
# Multiple R-squared:  0.7652,  Adjusted R-squared:  0.6956
# F-statistic:    11 on 8 and 27 DF,  p-value: 9.426e-07
```

We create the interaction plot next:

```
temp <- seq(from=-1, to=1, length.out=100)
y1 <- predict(model, data.frame('Temperature'=temp,
↪  'MaterialType'=rep('1', 100)))
y2 <- predict(model, data.frame('Temperature'=temp,
↪  'MaterialType'=rep('2', 100)))
y3 <- predict(model, data.frame('Temperature'=temp,
↪  'MaterialType'=rep('3', 100)))
plot(
    x=df$Temperature,
    y=df$BatteryLife,
    pch=19,
    col=c('orange','lightblue','green')[as.numeric(df$MaterialType)],
    xlab='Coded Temperature',
    ylab='Battery Life'
)
lines(x=temp, y=y1, col='orange')
lines(x=temp, y=y2, col='lightblue')
lines(x=temp, y=y3, col='green')
legend(
    'topright',
    legend=c('Material 1', 'Material 2', 'Material 3'),
    col=c('orange', 'lightblue', 'green'),
    lty=c(1, 1, 1)
)
```



Example 5.5

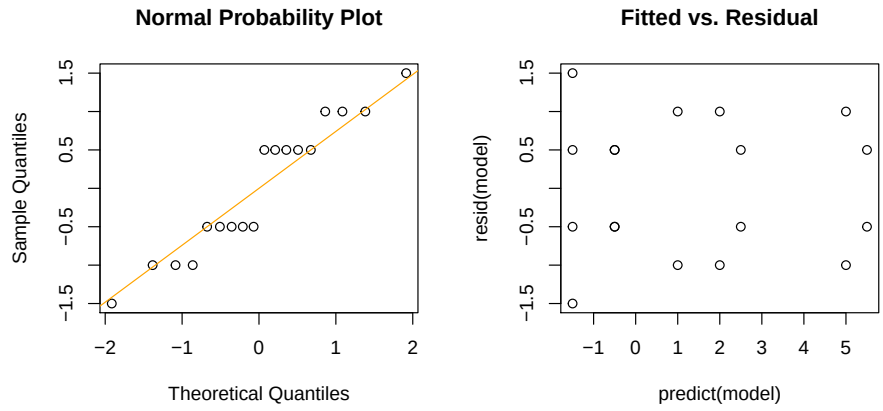
The tool life experiment of Example 5.5 examines how the life of a cutting tool is affected by the tool's cutting speed and the tool angle. We begin by fitting a regression model to produce the ANOVA table and coefficient estimates:

```
df <- Table5.16
A <- factor(df$TotalAngle)
B <- factor(df$CuttingSpeed)
model <- lm(ToolLife ~ A * B, data=df)
print(anova(model))
# Analysis of Variance Table
#
# Response: ToolLife
#           Df Sum Sq Mean Sq F value   Pr(>F)
# A          2 24.333 12.1667  8.4231 0.008676 **
# B          2 25.333 12.6667  8.7692 0.007703 **
# A:B        4 61.333 15.3333 10.6154 0.001844 **
# Residuals  9 13.000  1.4444
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
print(summary(model))
#
# Call:
# lm(formula = ToolLife ~ A * B, data = df)
#
# Residuals:
#    Min     1Q Median     3Q    Max
#   -1.5   -0.5    0.0    0.5    1.5
#
```

```
# Coefficients:
#               Estimate Std. Error t value Pr(>|t|)
# (Intercept)  1.333e+00  2.833e-01   4.707  0.00111 **
# A1          -1.500e+00  4.006e-01  -3.744  0.00460 **
# A2           1.333e+00  4.006e-01   3.328  0.00882 **
# B1          -1.667e+00  4.006e-01  -4.160  0.00245 **
# B2           6.667e-01  4.006e-01   1.664  0.13045
# A1:B1        3.333e-01  5.666e-01   0.588  0.57077
# A2:B1       -1.088e-15  5.666e-01   0.000  1.00000
# A1:B2       -2.000e+00  5.666e-01  -3.530  0.00641 **
# A2:B2       -1.333e+00  5.666e-01  -2.353  0.04306 *
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 1.202 on 9 degrees of freedom
# Multiple R-squared:  0.8952,  Adjusted R-squared:  0.802
# F-statistic: 9.606 on 8 and 9 DF,  p-value: 0.001337
```

Examining the diagnostic plots we don't find evidence of severe problems:

```
op <- par(mfrow=c(1,2))
qqnorm(resid(model), main='Normal Probability Plot')
qqline(resid(model), col='orange')
plot(x=predict(model), y=resid(model), main='Fitted vs. Residual')
```




```
par(op)
```

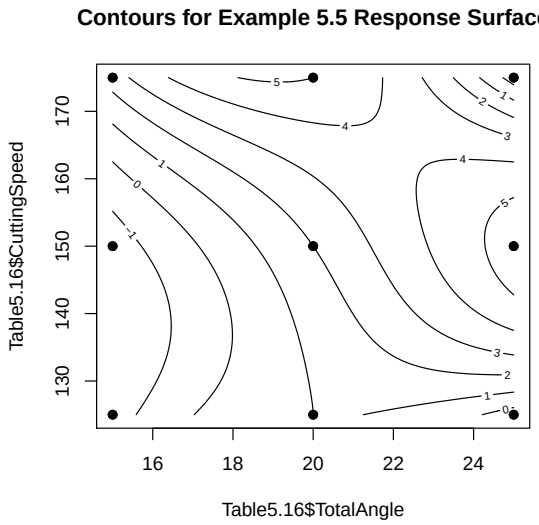
As the text states, since this model has quantitative factors at three levels each we could fit a second order model. First, let's create a new `data.frame` with the centered variables in it and then fit the model:

```
attach(Table5.16)
df <- data.frame(
    'ToolLife' = ToolLife,
    'Angle'=TotalAngle,
    'Speed'=CuttingSpeed,
    'Angle.Angle'=(TotalAngle - 20)^2,
    'Angle.Speed'=(TotalAngle - 20)*(CuttingSpeed-150),
    'Speed.Speed'=(CuttingSpeed-150)^2,
    'Angle.Angle.Speed'=(TotalAngle - 20)^2 * (CuttingSpeed-150),
    'Speed.Speed.Angle'=(CuttingSpeed-150)^2 * (TotalAngle - 20),
    'Angle.Speed.Angle.Speed'=(TotalAngle - 20)^2 *
    ↪  (CuttingSpeed-150)^2
)
detach(Table5.16)
model2 <- lm(ToolLife ~ ., data=df)
summary(model2)
#
# Call:
# lm(formula = ToolLife ~ ., data = df)
#
# Residuals:
#    Min     1Q Median     3Q    Max
#   -1.5   -0.5    0.0    0.5    1.5
#
# Coefficients:
#                           Estimate Std. Error t value Pr(>|t|)
# (Intercept)             -2.400e+01  4.416e+00  -5.435 0.000414 ***
# Angle                    7.000e-01  1.202e-01   5.824 0.000252 ***
# Speed                    8.000e-02  2.404e-02   3.328 0.008824 **
# Angle.Angle              1.778e-17  4.163e-02   0.000 1.000000
# Angle.Speed             -8.000e-03  3.399e-03  -2.353 0.043065 *
# Speed.Speed              1.600e-03  1.665e-03   0.961 0.361768
# Angle.Angle.Speed       -1.600e-03  1.178e-03  -1.359 0.207306
# Speed.Speed.Angle       -1.280e-03  2.355e-04  -5.435 0.000414 ***
# Angle.Speed.Angle.Speed -1.920e-04  8.158e-05  -2.353 0.043065 *
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 1.202 on 9 degrees of freedom
# Multiple R-squared:  0.8952,  Adjusted R-squared:  0.802
# F-statistic: 9.606 on 8 and 9 DF,  p-value: 0.001337
```

Next, we construct a contour plot of the surface, with the design points shown as dots:

```
plot(x=Table5.16$TotalAngle, y=Table5.16$CuttingSpeed, main='Contours
↪  for Example 5.5 Response Surface', pch=19)
x <- seq(from=min(Table5.16$TotalAngle), to=max(Table5.16$TotalAngle),
↪  length.out=100)
y <- seq(from=min(Table5.16$CuttingSpeed),
↪  to=max(Table5.16$CuttingSpeed), length.out=100)
```

```
df <- expand.grid('Angle'=x, 'Speed'=y)
df['Angle.Angle'] <- (df$Angle - 20)^2
df['Angle.Speed'] <- (df$Angle - 20) * (df$Speed - 150)
df['Speed.Speed'] <- (df$Speed - 150)^2
df['Angle.Angle.Speed'] <- (df$Angle - 20)^2 * (df$Speed - 150)
df['Speed.Speed.Angle'] <- (df$Speed - 150)^2 * (df$Angle - 20)
df['Angle.Speed.Angle.Speed'] <- (df$Angle - 20)^2 * (df$Speed -
↪  150)^2
z <- matrix(predict(model2, df), nrow=length(x))
contour(x=x, y=y, z=z, add=TRUE)
```



Compare this plot with Figure 5.19.

The `rgl` package (Adler, Murdoch, and others 2019) allows the creating of really slick 3-d visualizations with which you can directly interact. Try out the following code:

```
library(rgl)

open3d()
pallete <- topo.colors(255)
colors <- pallete[as.numeric(cut(as.numeric(z), 255))]
persp3d(
    x, y, z, col=colors,
    xlab='ToolAngle', ylab='CuttingSpeed', zlab='ToolLife')
```

Example 5.6

The experiment detailed in Table 5.24 from the textbook is an attempt to improve target detection on a radar scope over different amounts of background noise and "ground clutter". First, we model `Intensity` based on `GroundClutter` and `FilterType` (and their interaction) and block on `Operator`:

```
model <- aov(Intensity ~ Error(Operator) + GroundClutter * FilterType,
↪  data=Table5.21)
summary(model)
#
# Error: Operator
#           Df Sum Sq Mean Sq F value Pr(>F)
# Residuals  3  402.2   134.1
#
# Error: Within
#                           Df Sum Sq Mean Sq F value   Pr(>F)
# GroundClutter              2  335.6   167.8  15.132 0.000253 ***
# FilterType                 1 1066.7  1066.7  96.192 6.45e-08 ***
# GroundClutter:FilterType   2   77.1    38.5   3.476 0.057507 .
# Residuals                 15  166.3    11.1
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Next, let's model these as random effects:

```
library(lme4)
library(lmerTest)
model <- lmer(
    Intensity ~ GroundClutter * FilterType + (1|Operator),
    REML=TRUE,
    data=Table5.21
)
print(summary(model))
# Linear mixed model fit by REML. t-tests use Satterthwaite's method [
# lmerModLmerTest]
# Formula: Intensity ~ GroundClutter * FilterType + (1 | Operator)
#    Data: Table5.21
#
# REML criterion at convergence: 118.7
#
# Scaled residuals:
#     Min      1Q  Median      3Q     Max
# -1.4661 -0.4704 -0.0127  0.6198  1.6870
#
# Random effects:
#  Groups   Name        Variance Std.Dev.
#  Operator (Intercept) 20.49    4.527
```

```
#  Residual             11.09    3.330
# Number of obs: 24, groups:  Operator, 4
#
# Fixed effects:
#                            Estimate Std. Error      df t value
↪  Pr(>|t|)
# (Intercept)                 94.9167     2.3634  3.0000  40.161
↪  3.40e-05
# GroundClutter1               4.3333     0.9613 15.0000   4.508
↪  0.000417
# GroundClutter2              -4.7917     0.9613 15.0000  -4.985
↪  0.000163
# FilterType1                  6.6667     0.6797 15.0000   9.808
↪  6.45e-08
# GroundClutter1:FilterType1   2.0833     0.9613 15.0000   2.167
↪  0.046722
# GroundClutter2:FilterType1  -2.2917     0.9613 15.0000  -2.384
↪  0.030774
#
# (Intercept)                *
# GroundClutter1             *
# GroundClutter2             *
# FilterType1                *
# GroundClutter1:FilterType1 *
# GroundClutter2:FilterType1 *
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Correlation of Fixed Effects:
#             (Intr) GrndC1 GrndC2 FltrT1 GC1:FT
# GrondClttr1  0.000
# GrondClttr2  0.000 -0.500
# FilterType1  0.000  0.000  0.000
# GrndCl1:FT1  0.000  0.000  0.000  0.000
# GrndCl2:FT1  0.000  0.000  0.000  0.000 -0.500
print(rand(model))
# ANOVA-like table for random-effects: Single term deletions
#
# Model:
# Intensity ~ GroundClutter + FilterType + (1 | Operator) +
↪  GroundClutter:FilterType
#                npar  logLik    AIC    LRT Df Pr(>Chisq)
# <none>            8 -59.368 134.74
# (1 | Operator)    7 -65.050 144.10 11.363  1   0.000749 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Section 5.6 Radar Detection as a Latin Square Design

Later in Section 5.6, the analysis is run as a 3×2 Latin Square design. Thehe result of this analysis is given in Table 5.25. Here, we modify our fixed-effects approach by adding `Day` into the list of error terms for blocking:

```
model <- aov(Intensity ~ Error(Day + Operator) + GroundClutter *
↪  Filter, data=Table5.24)
summary(model)
#
# Error: Day
#           Df Sum Sq Mean Sq F value Pr(>F)
# Residuals  5  4.333  0.8667
#
# Error: Operator
#           Df Sum Sq Mean Sq F value Pr(>F)
# Residuals  5    428    85.6
#
# Error: Within
#                      Df Sum Sq Mean Sq F value   Pr(>F)
# GroundClutter         2  571.5   285.8   28.86 1.27e-06 ***
# Filter                1 1469.4  1469.4  148.43 1.04e-10 ***
# GroundClutter:Filter  2  126.7    63.4    6.40   0.0071 **
# Residuals            20  198.0     9.9
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

References

Adler, Daniel, Duncan Murdoch, and others. 2019. *Rgl: 3D Visualization Using Opengl*. https://cran.r-project.org/package=rgl.

Brien, Chris. 2018. *Dae: Functions Useful in the Design and Anova of Experiments*. https://cran.r-project.org/package=dae.

Chapter 6

Two-Level Factorial Designs

We can easily construct any factorial using the `expand.grid` function from R:

```
expand.grid('A'=c(-1,1), 'B'=c(-1, 1), 'C'=c(-1, 1))
#    A  B  C
# 1 -1 -1 -1
# 2  1 -1 -1
# 3 -1  1 -1
# 4  1  1 -1
# 5 -1 -1  1
# 6  1 -1  1
# 7 -1  1  1
# 8  1  1  1
```

For an arbitrary k we can use `lapply` to return the correct number of `c(-1,1)` values, and then use `do.call` to use those as an argument for `expand.grid`:

```
fac2 <- function(k){
    as.data.frame(do.call(
        expand.grid,
        lapply(
            1:k,
            function(.){
                c(-1,1)
            }
        )
    ))
}

fac2(4)
#    Var1 Var2 Var3 Var4
# 1    -1   -1   -1   -1
# 2     1   -1   -1   -1
# 3    -1    1   -1   -1
# 4     1    1   -1   -1
# 5    -1   -1    1   -1
# 6     1   -1    1   -1
# 7    -1    1    1   -1
```

```
# 8     1    1    1   -1
# 9    -1   -1   -1    1
# 10    1   -1   -1    1
# 11   -1    1   -1    1
# 12    1    1   -1    1
# 13   -1   -1    1    1
# 14    1   -1    1    1
# 15   -1    1    1    1
# 16    1    1    1    1
```

Section 6.2

Section 6.2 describes an experiment where the experimenter seeks to optimize yield by varying reactant concentration (`A`) and catalyst amount (`B`). I transcribed the data given in Figure 6.1.

```
str(Figure6.1)
# 'data.frame': 12 obs. of  3 variables:
#  $ A    : num  -1 1 -1 1 -1 1 -1 1 -1 1 ...
#  $ B    : num  -1 -1 1 1 -1 -1 1 1 -1 -1 ...
#  $ Yield: num  28 36 18 31 25 32 19 30 27 32 ...
```

This produces the following model:

```
model <- aov(Yield ~ A*B, data=Figure6.1)
summary(model)
#             Df Sum Sq Mean Sq F value   Pr(>F)
# A            1 208.33  208.33  53.191 8.44e-05 ***
# B            1  75.00   75.00  19.149  0.00236 **
# A:B          1   8.33    8.33   2.128  0.18278
# Residuals    8  31.33    3.92
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

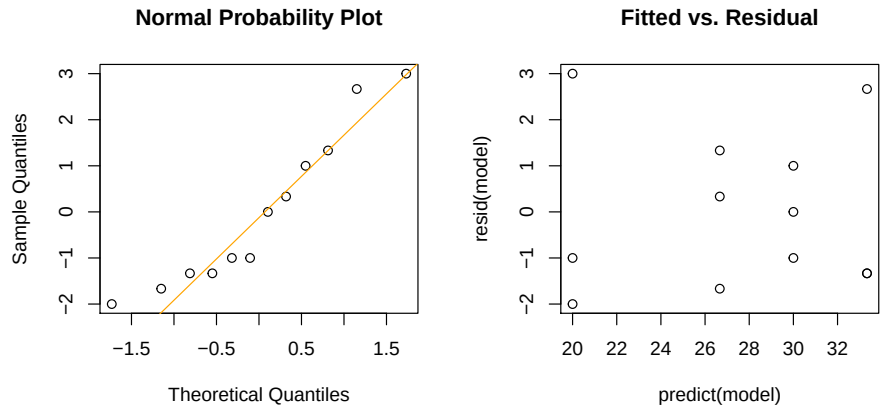
Table 6.2 gives the signs for calculating the effects. Here we replicate 3 times as in Figure 6.1. The `model.matrix` function shows us what is actually being fit by the linear model:

```
model.matrix(~A*B, data=Figure6.1)
#    (Intercept)  A  B A:B
# 1            1 -1 -1   1
# 2            1  1 -1  -1
```

```
# 3            1 -1  1  -1
# 4            1  1  1   1
# 5            1 -1 -1   1
# 6            1  1 -1  -1
# 7            1 -1  1  -1
# 8            1  1  1   1
# 9            1 -1 -1   1
# 10           1  1 -1  -1
# 11           1 -1  1  -1
# 12           1  1  1   1
# attr(,"assign")
# [1] 0 1 2 3
```

Diagnostic plots for the model are given below:

```
model <- lm(Yield ~ A*B, data=Figure6.1)

op <- par(mfrow=c(1,2))
qqnorm(resid(model), main='Normal Probability Plot')
qqline(resid(model), col='orange')
plot(x=predict(model), y=resid(model), main='Fitted vs. Residual')
```




```
par(op)
```

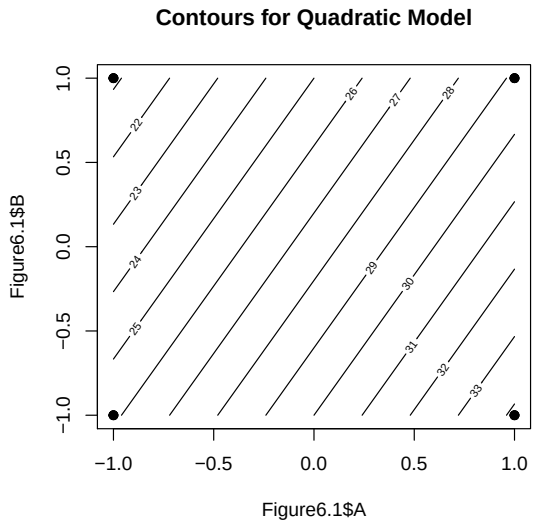
Figure 6.3 in the text contains a contour plot of the response surface. First, let's model this as linear regression. Note that the text uses a *first order* model so it only contains main effects.

```
model <- lm(Yield ~ A + B, data=Figure6.1)
summary(model)
#
```

```
# Call:
# lm(formula = Yield ~ A + B, data = Figure6.1)
#
# Residuals:
#     Min      1Q  Median      3Q     Max
# -2.8333 -1.9167  0.3333  1.8333  2.1667
#
# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept)   27.500      0.606  45.377 6.13e-12 ***
# A              4.167      0.606   6.875 7.27e-05 ***
# B             -2.500      0.606  -4.125  0.00258 **
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 2.099 on 9 degrees of freedom
# Multiple R-squared:  0.8772,  Adjusted R-squared:  0.8499
# F-statistic: 32.14 on 2 and 9 DF,  p-value: 7.971e-05
```

Next, we plot the design points and show the contour plot:

```
plot(x=Figure6.1$A, y=Figure6.1$B, main='Contours for Quadratic
↪  Model', pch=19)
x <- seq(from=-1, to=1, length.out=100)
y <- seq(from=-1, to=1, length.out=100)
df <- expand.grid('A'=x, 'B'=y)
z <- matrix(predict(model, df), nrow=length(x))
contour(x=x, y=y, z=z, add=TRUE)
```



Example 6.1 — Plasma Etching

A 2^3 factorial design was executed to observe the effect of the gap between electrodes, gas flow, and RF-power on a plasma etching tool's etch rate. The data is in Table 6.4. Compare the ANOVA table of Table 6.6 to the following:

```
model <- aov(EtchRate ~ Gap * Flow * Power, data=Table6.4)
summary(model)
#                Df Sum Sq Mean Sq F value   Pr(>F)
# Gap             1  41311   41311  18.339 0.002679 **
# Flow            1    218     218   0.097 0.763911
# Power           1 374850  374850 166.411 1.23e-06 ***
# Gap:Flow        1   2475    2475   1.099 0.325168
# Gap:Power       1  94403   94403  41.909 0.000193 ***
# Flow:Power      1     18      18   0.008 0.930849
# Gap:Flow:Power  1    127     127   0.056 0.818586
# Residuals       8  18020    2253
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The example also gives effect estimates. We can find the coefficients using the `coef` function. Recall that the effect size is twice the parameter estimate, so the effects are:

```
2*coef(model)[-1]
#            Gap           Flow          Power       Gap:Flow
↪  Gap:Power
#       -101.625          7.375        306.125        -24.875
↪  -153.625
#     Flow:Power Gap:Flow:Power
#         -2.125          5.625
```

Example 6.2 — Filtration Rate

In Example 6.2 the filtration rate of a chemical process is studied through a 2^4 factorial design varying temperature, pressure, concentration of formaldehyde, and stirring rate. Since this is a saturated design we don't have an estimate of error and can't use the regular `aov` procedure. Doing so results simply in the partition of the sums of squares:

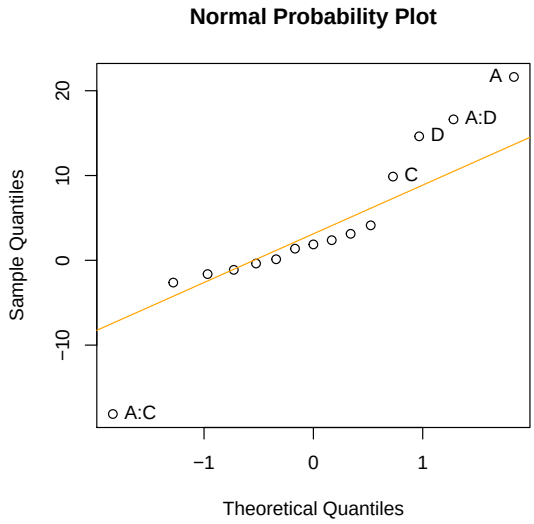
```
model <- aov(Filtration ~ Temperature * Pressure * Formaldehyde *
↪  StirringRate, data=Table6.10)
summary(model)
#                                                 Df Sum Sq Mean Sq
# Temperature                                      1 1870.6  1870.6
# Pressure                                         1   39.1    39.1
# Formaldehyde                                     1  390.1   390.1
# StirringRate                                     1  855.6   855.6
# Temperature:Pressure                             1    0.1     0.1
# Temperature:Formaldehyde                         1 1314.1  1314.1
# Pressure:Formaldehyde                            1   22.6    22.6
# Temperature:StirringRate                         1 1105.6  1105.6
# Pressure:StirringRate                            1    0.6     0.6
# Formaldehyde:StirringRate                        1    5.1     5.1
# Temperature:Pressure:Formaldehyde                1   14.1    14.1
# Temperature:Pressure:StirringRate                1   68.1    68.1
# Temperature:Formaldehyde:StirringRate            1   10.6    10.6
# Pressure:Formaldehyde:StirringRate               1   27.6    27.6
# Temperature:Pressure:Formaldehyde:StirringRate   1    7.6     7.6
```

Instead we can create a normal probability plot based on the effects:

```
df <- Table6.10
colnames(df) <- c('A', 'B', 'C', 'D', 'Filtration', 'Block')
X <- model.matrix(~ -1 + A*B*C*D, data=df)
effects <- apply(X, 2, function(w){ sum((w*df$Filtration) / (0.5 *
↪  dim(df)[1])) })
names(effects) <- colnames(X)
effects
#       A       B       C       D     A:B     A:C     B:C     A:D
↪  B:D
#  21.625   3.125   9.875  14.625   0.125 -18.125   2.375  16.625
↪  -0.375
#     C:D   A:B:C   A:B:D   A:C:D   B:C:D A:B:C:D
#  -1.125   1.875   4.125  -1.625  -2.625   1.375
```

Plotting these we get:

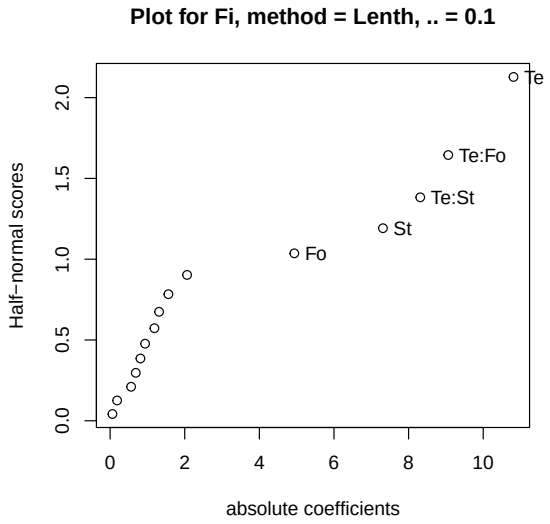
```
locs <- qqnorm(effects, main='Normal Probability Plot')
qqline(effects, col='orange')
ix <- c(1,3,4,6,8)
text(x=locs$x[ix], y=locs$y[ix], labels=names(effects[ix]), pos=c(2,
↪  4, 4, 4, 4))
```



If you compare this plot to the output in Figure 6.11 you'll note that the line in the text better traces the central points than R's `qqline`. This is because the `qqline` procedure is geared towards an overall normality assumption, whereas the line in Figure 6.11 is using a method that focuses on the interior points for establishing the variance of the distribution.

When the design is saturated, i.e. when there are no degrees of freedom for error, then the `halfnormal` function from package `DoE.base` (Groemping, Amarov, and Xu 2019) will construct a normal probability plot and label effects based on Lenth's pseudo-p-value approach:

```
library(DoE.base)

Fo <- Table6.10$Formaldehyde
Te <- Table6.10$Temperature
St <- Table6.10$StirringRate
Pr <- Table6.10$Pressure
Fi <- Table6.10$Filtration
model <- aov(Fi ~ Te * Pr * Fo * St, data=Table6.10)
vals <- halfnormal(model, alpha=0.1)
```

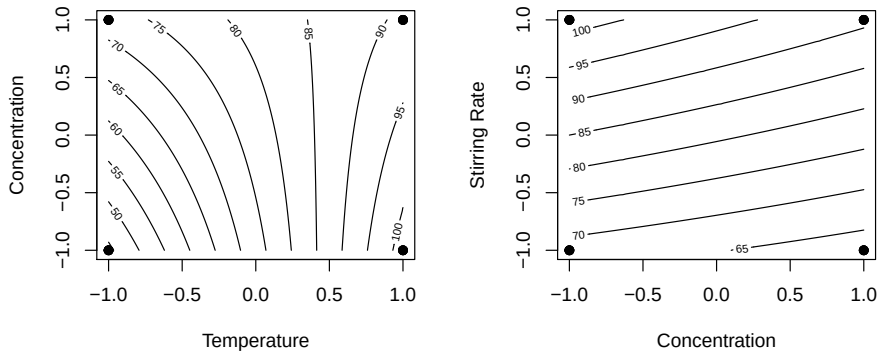


The returned object from `halfnormal` is a list with `signif` containing those factors that were significant at the α given:

```
print(vals$signif)
# [1] "Fo"    "St"    "Te:St" "Te:Fo" "Te"
```

Next, we construct the contour plots of Figure 6.14:

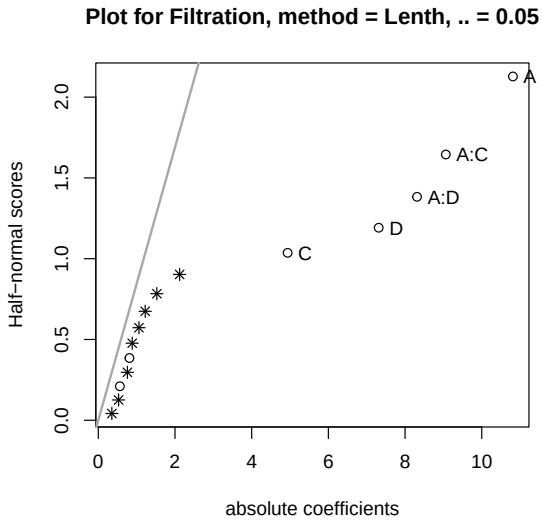
```
op <- par(mfrow=c(1,2))
model <- lm(Filtration ~ A * C * D, data=df)

# Temperature (A) vs. Concentration (C)
plot(x=df$A, y=df$C, pch=19, xlab='Temperature', ylab='Concentration')
x <- seq(from=-1, to=1, length.out=100)
y <- seq(from=-1, to=1, length.out=100)
pre.z <- expand.grid('A'=x, 'C'=y)
pre.z[,'D'] <- rep(1, 100)
z <- matrix(predict(model, pre.z), nrow=length(x))
contour(x=x, y=y, z=z, add=TRUE)

# Concentration (C) vs. Stirring Rate (D)
plot(x=df$C, y=df$D, pch=19, xlab='Concentration', ylab='Stirring
↪  Rate')
x <- seq(from=-1, to=1, length.out=100)
y <- seq(from=-1, to=1, length.out=100)
pre.z <- expand.grid('C'=x, 'D'=y)
pre.z[,'A'] <- rep(1, 100)
z <- matrix(predict(model, pre.z), nrow=length(x))
contour(x=x, y=y, z=z, add=TRUE)
```




```
par(op)
```

Compare Figure 6.18, where an outlier is removed, to the the following:

```
model <- aov(Filtration ~ A * C * D, data=df)
vals <- halfnormal(model)
```

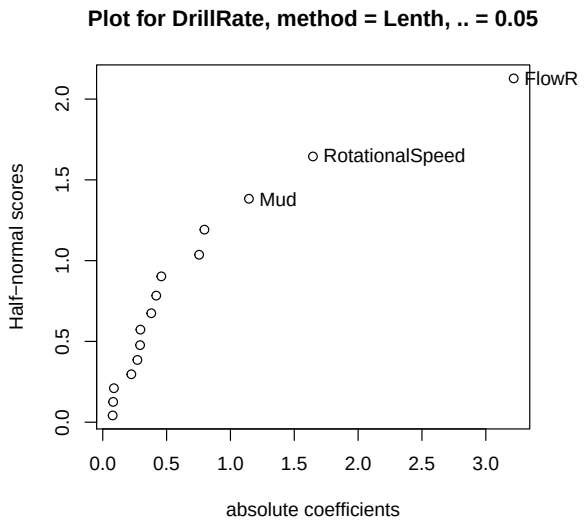



```
print(vals$signif)
# [1] "C"   "D"   "A:D" "A:C" "A"
```

Example 6.3 — Data Transformation in a Factorial Design

Example 6.3 describes an experiment where a 2^4 factorial design was used to study the advance rate of a drill as a function of load, flow rate, rotation speed, and the type of drilling mud being used. The data is stored in `Example6.3`. First, we fit a regression model and then examine the half-normal probability plot:

```
model <- lm(DrillRate ~ .*.*.*., data=Example6.3)
vals <- halfnormal(model)
```




```
print(vals$signif)
# [1] "Mud"             "RotationalSpeed" "FlowRate"
```

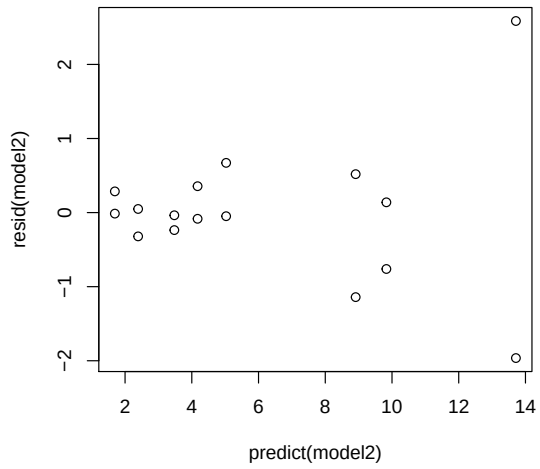
With `FlowRate`, `RotationalSpeed`, and `Mud` main effects all appearing significant, we examine the 2^3 factorial replicated twice that's left when we ignore `Load`:

```
model2 <- aov(DrillRate ~ Mud + RotationalSpeed + FlowRate +
↪  FlowRate:Mud + FlowRate:RotationalSpeed, data=Example6.3)
summary(model2)
#                          Df Sum Sq Mean Sq F value   Pr(>F)
# Mud                       1  20.98   20.98  15.484  0.00280 **
```

```
# RotationalSpeed           1  43.36   43.36  32.009  0.00021 ***
# FlowRate                  1 165.77  165.77 122.363 6.26e-07 ***
# Mud:FlowRate              1  10.14   10.14   7.488  0.02096 *
# RotationalSpeed:FlowRate  1   9.12    9.12   6.732  0.02673 *
# Residuals                10  13.55    1.35
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

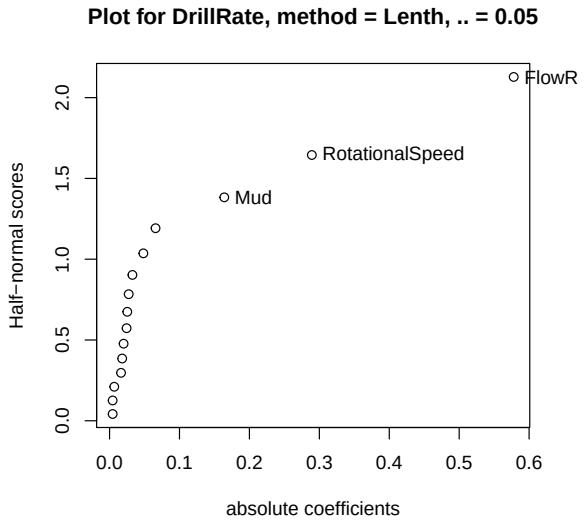
Next, let's examine the predicted versus residual plot:

```
plot(predict(model2), resid(model2))
```



The diagnostic plot shows severely unequal variance. As the text suggests, let's look at the logarithm of `DrillRate`:

```
df <- Example6.3
df[,'DrillRate'] <- log(df[,'DrillRate'])
model3 <- lm(DrillRate ~ .*.*.*., data=df)
vals <- halfnormal(model3)
```




```
print(vals$signif)
# [1] "Mud"              "RotationalSpeed" "FlowRate"
```

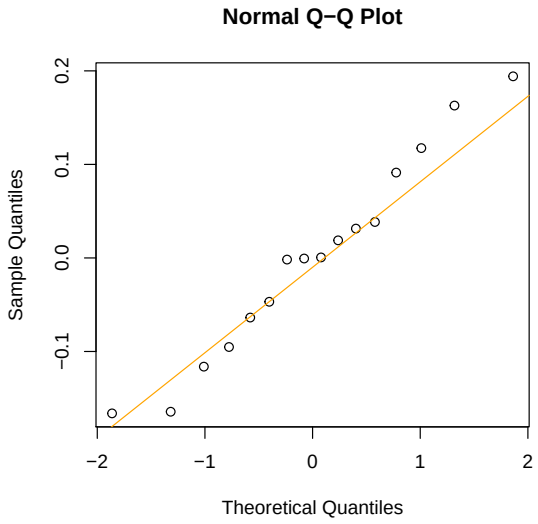
Again we find the three main effects to be active. We examine the ANOVA table of the reduced model and find that none of the interaction effects seem to be active:

```
model4 <- lm(DrillRate ~ FlowRate * RotationalSpeed * Mud, data=df)
anova(model4)
# Analysis of Variance Table
#
# Response: DrillRate
#                              Df Sum Sq Mean Sq  F value    Pr(>F)
↪
# FlowRate                      1 5.3452  5.3452 356.5307 6.399e-08
↪  ***
# RotationalSpeed               1 1.3389  1.3389  89.3040 1.293e-05
↪  ***
# Mud                           1 0.4305  0.4305  28.7173 0.0006786
↪  ***
# FlowRate:RotationalSpeed      1 0.0095  0.0095   0.6320 0.4495577
↪
# FlowRate:Mud                  1 0.0373  0.0373   2.4901 0.1532191
↪
# RotationalSpeed:Mud           1 0.0007  0.0007   0.0479 0.8321670
↪
# FlowRate:RotationalSpeed:Mud  1 0.0052  0.0052   0.3467 0.5722503
↪
# Residuals                     8 0.1199  0.0150
↪
```

```
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

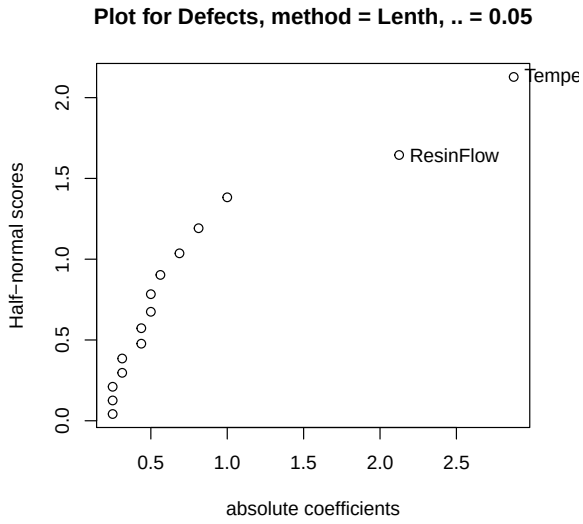
We fit the main-effects-only model and check for normality of the residuals:

```
model4 <- lm(DrillRate ~ FlowRate + RotationalSpeed + Mud, data=df)
qqnorm(resid(model4))
qqline(resid(model4), col='orange')
```



Example 6.4 — Location and Dispersion Effects in an Unreplicated Factorial

The experiment in Example 6.4 seeks to minimize the number of defects by controlling the temperature, clamp time, resin flow, and closing time in a panel extruder. There's two ways to improve the panel quality. First, we can decrease the mean number of defects. Second, we can reduce the variability of the number of defects. We begin with the mean defects:



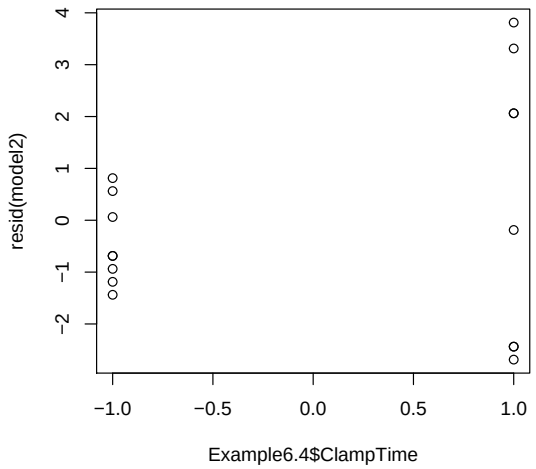

```
# [1] "ResinFlow"   "Temperature"
```

Only two of the main effects are active, so we examine the projection of the original 2^4 design down to a 2^2.

```
model2 <- lm(Defects ~ Temperature * ResinFlow, data=Example6.4)
anova(model2)
# Analysis of Variance Table
#
# Response: Defects
#                      Df  Sum Sq Mean Sq F value   Pr(>F)
# Temperature           1 132.250 132.250  27.421 0.000209 ***
# ResinFlow             1  72.250  72.250  14.981 0.002226 **
# Temperature:ResinFlow  1   1.563   1.563   0.324 0.579727
# Residuals            12  57.875   4.823
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

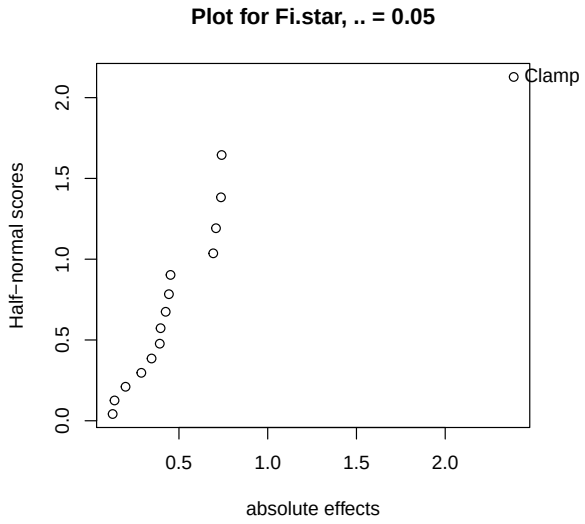
The interaction effect is not significant so we proceed with the main-effects only model. We present the residual as a function of clamp-time below:

```
model2 <- lm(Defects ~ Temperature + ResinFlow, data=Example6.4)
plot(Example6.4$ClampTime, resid(model2))
```



This suggests that the assumption of equal variance is incorrect, but also that minimizing `ClampTime` could reduce variability. Let us examine this idea further by calculating the dispersion effect in Example 6.4:

```
ix <- Example6.4$ClampTime == 1
X <- model.matrix(Defects ~ -1 + . * . * . * ., data=Example6.4)
Fi.star <- apply(
    X,
    2,
    function(w){
        log(
            sd(resid(model2)[w == 1])^2
            /
            sd(resid(model2)[w != 1])^2
        )
    }
)
vals <- halfnormal(Fi.star)
```



Indeed, `ClampTime` has a strong effect on variability.

Example 6.5 — Duplicate Measurements on the Response

In this experiment engineers stacked four wafers in a vertical oxidation furnace and examined the effect of temperature, time, pressure, and gas flow on the oxide thinness of the wafer. Note that since the wafers were all exposed at the same time they don't count as separate runs and are instead examples of repeated measurements. We begin by examining the sample variance of the wafers in a run:

```
df <- aggregate(OxideThickness ~ ., data=Table6.18, FUN=mean)
df['SampleVariance'] <- aggregate(OxideThickness ~ ., data=Table6.18,
↪  FUN=var)['OxideThickness']
```

Temperature	Time	Pressure	GasFlow	OxideThickness	SampleVariance
-1	-1	-1	-1	378	2.0000000
1	-1	-1	-1	416	0.6666667
-1	1	-1	-1	381	3.3333333
1	1	-1	-1	448	3.3333333
-1	-1	1	-1	372	6.6666667
1	-1	1	-1	390	2.0000000
-1	1	1	-1	385	0.6666667
1	1	1	-1	430	8.6666667
-1	-1	-1	1	380	12.0000000
1	-1	-1	1	415	14.6666667
-1	1	-1	1	371	0.6666667
1	1	-1	1	446	6.0000000
-1	-1	1	1	378	1.3333333
1	-1	1	1	392	34.0000000
-1	1	1	1	376	0.6666667
1	1	1	1	429	1.3333333

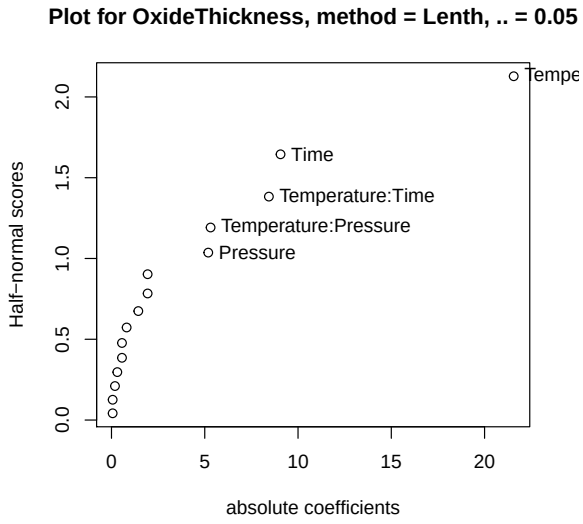
Next, we partition the effects and sums of squares for the mean oxide thickness over the four wafers:

```
model <- lm(OxideThickness ~ Temperature * Time * Pressure * GasFlow,
↪  data=df)
model.ss <- anova(model)[['Sum Sq']]
effect.estimates <- data.frame(
    'EffectEstimate'=2 * coef(model)[-1],
    'SumOfSquares'=model.ss[-1],
    'Contribution'=sprintf('%0.2f%%',
    ↪  100*model.ss[-1]/sum(model.ss[-1]))
)
```

	EffectEstimate	SumOfSquares	Contribution
Temperature	43.125	1314.0625	37.42%
Time	18.125	430.5625	12.26%
Pressure	-10.375	10.5625	0.30%
GasFlow	-1.625	1139.0625	32.44%
Temperature:Time	16.875	451.5625	12.86%
Temperature:Pressure	-10.625	60.0625	1.71%
Time:Pressure	3.875	5.0625	0.14%
Temperature:GasFlow	1.125	60.0625	1.71%
Time:GasFlow	-3.875	5.0625	0.14%
Pressure:GasFlow	1.125	0.5625	0.02%
Temperature:Time:Pressure	-0.375	33.0625	0.94%
Temperature:Time:GasFlow	2.875	0.0625	0.00%
Temperature:Pressure:GasFlow	-0.125	1.5625	0.04%
Time:Pressure:GasFlow	-0.625	0.0625	0.00%
Temperature:Time:Pressure:GasFlow	0.125	0.0000	0.00%

Examining the half-normal probability plot we find `Temperature`, `Time`, `Pressure`, and the interactions of `Temperature` with `Time` and `Pressure` to be significant:

```
vals <- halfnormal(model)
```




```
print(vals$signif)
# [1] "Pressure"              "Temperature:Pressure" "Temperature:Time"
↪
# [4] "Time"                  "Temperature"
```

Fitting this reduced model shows good significance:

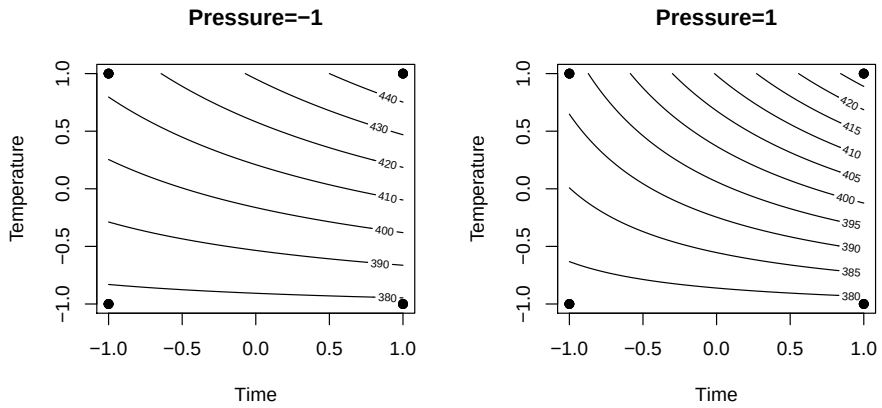
```
model2 <- lm(OxideThickness ~ Temperature + Time + Temperature:Time +
↪  Pressure + Temperature:Pressure, data=df)
anova(model2)
# Analysis of Variance Table
#
# Response: OxideThickness
#                      Df Sum Sq Mean Sq F value    Pr(>F)
# Temperature           1 7439.1  7439.1 422.374 1.644e-09 ***
# Time                  1 1314.1  1314.1  74.610 5.978e-06 ***
# Pressure              1  430.6   430.6  24.446 0.0005834 ***
# Temperature:Time      1 1139.1  1139.1  64.674 1.125e-05 ***
# Temperature:Pressure  1  451.6   451.6  25.639 0.0004895 ***
# Residuals            10  176.1    17.6
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The parameter estimates are given next:

```
summary(model2)
#
# Call:
# lm.default(formula = OxideThickness ~ Temperature + Time +
↪  Temperature:Time +
#     Pressure + Temperature:Pressure, data = df)
#
# Residuals:
#    Min     1Q Median     3Q    Max
# -7.125 -2.469  1.000  2.250  6.625
#
# Coefficients:
#                      Estimate Std. Error t value Pr(>|t|)
# (Intercept)           399.188      1.049 380.475  < 2e-16 ***
# Temperature            21.562      1.049  20.552 1.64e-09 ***
# Time                    9.062      1.049   8.638 5.98e-06 ***
# Pressure               -5.187      1.049  -4.944 0.000583 ***
# Temperature:Time        8.438      1.049   8.042 1.12e-05 ***
# Temperature:Pressure   -5.313      1.049  -5.063 0.000489 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 4.197 on 10 degrees of freedom
# Multiple R-squared:  0.9839,  Adjusted R-squared:  0.9759
# F-statistic: 122.3 on 5 and 10 DF,  p-value: 1.237e-08
```

Next, we plot the response surface:

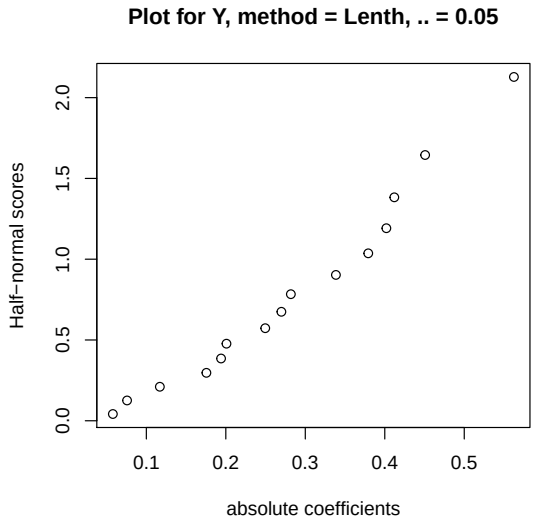
```
op <- par(mfrow=c(1,2))

plot(x=df$Time, y=df$Temperature, pch=19, xlab='Time',
↪  ylab='Temperature', main='Pressure=-1')
x <- seq(from=-1, to=1, length.out=100)
y <- seq(from=-1, to=1, length.out=100)
pre.z <- expand.grid('Time'=x, 'Temperature'=y)
pre.z[,'Pressure'] <- rep(-1, 100)
z <- matrix(predict(model2, pre.z), nrow=length(x))
contour(x=x, y=y, z=z, add=TRUE)

plot(x=df$Time, y=df$Temperature, pch=19, xlab='Time',
↪  ylab='Temperature', main='Pressure=1')
x <- seq(from=-1, to=1, length.out=100)
y <- seq(from=-1, to=1, length.out=100)
pre.z <- expand.grid('Time'=x, 'Temperature'=y)
pre.z[,'Pressure'] <- rep(1, 100)
z <- matrix(predict(model2, pre.z), nrow=length(x))
contour(x=x, y=y, z=z, add=TRUE)
```




```
par(op)
```

Consider now the log transform of the sample variance at each set of four wafers:

```
Y <- log(df[,'SampleVariance'])
model <- lm(Y ~ Temperature * Time * Pressure * GasFlow, data=df)
vals <- halfnormal(model)
```



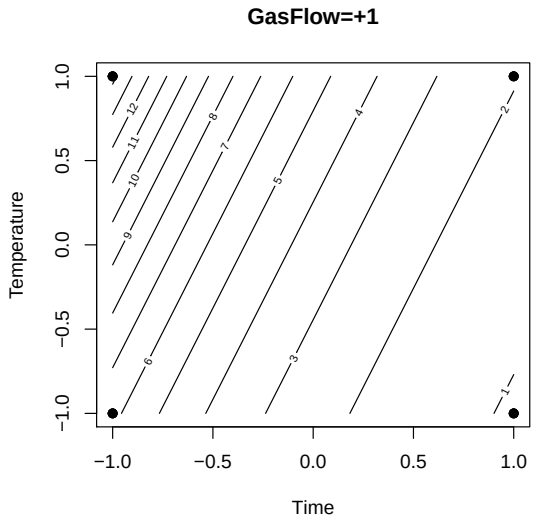

```
print(vals$signif)
# character(0)
```

The text notes there are no strong effects but that factor `A` and `B:D` are largest, so we fit that model:

```
model <- lm(Y ~ Temperature + Time + GasFlow + Time:GasFlow, data=df)
summary(model)
#
# Call:
# lm.default(formula = Y ~ Temperature + Time + GasFlow +
↪  Time:GasFlow,
#     data = df)
#
# Residuals:
#     Min      1Q  Median      3Q     Max
# -1.5464 -0.4391 -0.1105  0.6648  1.5896
#
# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept)    1.0808     0.2641   4.092  0.00178 **
# Temperature    0.4120     0.2641   1.560  0.14708
# Time          -0.4020     0.2641  -1.522  0.15623
# GasFlow        0.2008     0.2641   0.760  0.46305
# Time:GasFlow  -0.5625     0.2641  -2.130  0.05661 .
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 1.057 on 11 degrees of freedom
# Multiple R-squared:  0.4728,  Adjusted R-squared:  0.281
# F-statistic: 2.466 on 4 and 11 DF,  p-value: 0.1065
```

Next, we present the contour plot of this surface:

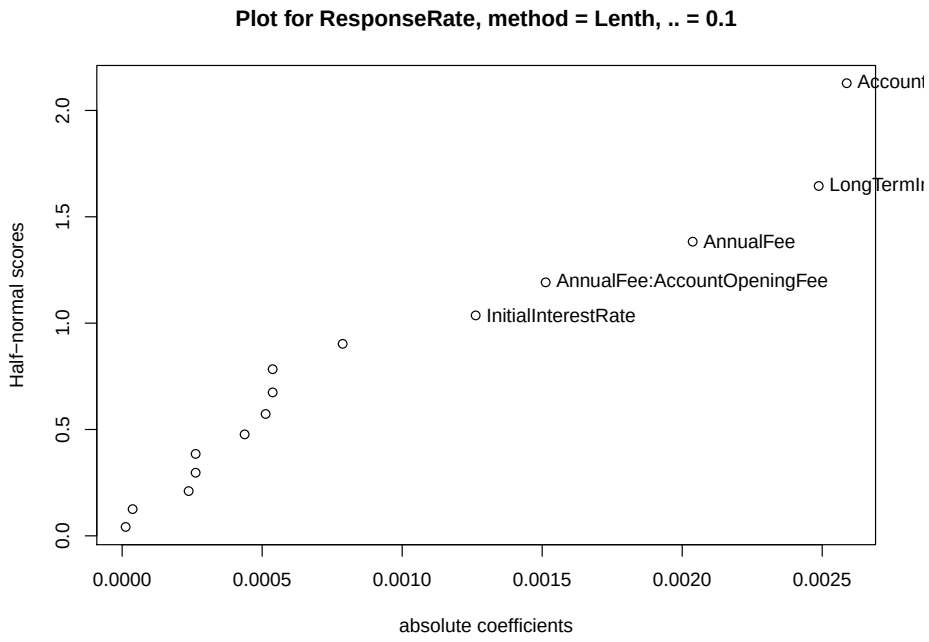
```
plot(x=df$Time, y=df$Temperature, pch=19, xlab='Time',
↪  ylab='Temperature', main='GasFlow=+1')
x <- seq(from=-1, to=1, length.out=100)
y <- seq(from=-1, to=1, length.out=100)
pre.z <- expand.grid('Time'=x, 'Temperature'=y)
pre.z[,'GasFlow'] <- rep(1, 100)
z <- matrix(predict(model, pre.z), nrow=length(x))
contour(x=x, y=y, z=exp(z), add=TRUE)
```



Example 6.6 — Credit Card Marketing

An experiment to increase direct mail sales was undertaken by a financial services company. They examined varying the annual fee, the account-opening fee, the initial interest rate, and the long term interest rate in a 2^4 design. The half-normal plot of effects is presented next:

```
model <- lm(ResponseRate ~ .*.*.*., data=Table6.22)
vals <- halfnormal(model, alpha=0.1)
```




```
print(vals$signif)
# [1] "InitialInterestRate"          "AnnualFee:AccountOpeningFee"
# [3] "AnnualFee"                    "LongTermInterestRate"
# [5] "AccountOpeningFee"
```

Putting these active effects into a regression model we find the coefficient estimates:

```
model <- lm(
    ResponseRate ~ AnnualFee + AccountOpeningFee + InitialInterestRate
↪  + LongTermInterestRate + AnnualFee:AccountOpeningFee,
    data=Table6.22
)
summary(model)
#
# Call:
# lm.default(formula = ResponseRate ~ AnnualFee + AccountOpeningFee +
#     InitialInterestRate + LongTermInterestRate +
↪  AnnualFee:AccountOpeningFee,
#     data = Table6.22)
#
# Residuals:
#        Min         1Q     Median         3Q        Max
# -0.0036250 -0.0006250  0.0001125  0.0007062  0.0026000
#
# Coefficients:
#                                Estimate Std. Error t value Pr(>|t|)
↪
```

```
# (Intercept)                  0.0236375  0.0004299  54.989 9.59e-14
↪  ***
# AnnualFee                    0.0020375  0.0004299   4.740 0.000792
↪  ***
# AccountOpeningFee           -0.0025875  0.0004299  -6.019 0.000129
↪  ***
# InitialInterestRate          0.0012625  0.0004299   2.937 0.014862 *
↪
# LongTermInterestRate        -0.0024875  0.0004299  -5.787 0.000176
↪  ***
# AnnualFee:AccountOpeningFee -0.0015125  0.0004299  -3.519 0.005552
↪  **
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 0.001719 on 10 degrees of freedom
# Multiple R-squared:  0.9188,  Adjusted R-squared:  0.8782
# F-statistic: 22.64 on 5 and 10 DF,  p-value: 3.722e-05
```

From this we could calculate the best combination.

Example 6.7

Recall the experiment in Example 6.2. In this example we need to add some center runs:

```
str(Table6.10)
# 'data.frame': 16 obs. of  6 variables:
#  $ Temperature : num  -1 1 -1 1 -1 1 -1 1 -1 1 ...
#  $ Pressure    : num  -1 -1 1 1 -1 -1 1 1 -1 -1 ...
#  $ Formaldehyde: num  -1 -1 -1 -1 1 1 1 1 -1 -1 ...
#  $ StirringRate: num  -1 -1 -1 -1 -1 -1 -1 -1 1 1 ...
#  $ Filtration  : num  45 71 48 65 68 60 80 65 43 100 ...
#  $ Block       : Factor w/ 2 levels "1","2": 1 2 2 1 2 1 1 2 2 1 ...
```

We add `NA` for the `Block` level. Adding the center runs given in the example:

```
Example6.7 <- rbind(
    Table6.10,
    c(0, 0, 0, 0, 73, NA),
    c(0, 0, 0, 0, 75, NA),
    c(0, 0, 0, 0, 66, NA),
    c(0, 0, 0, 0, 69, NA)
)
model1 <- lm(Filtration ~
↪  Temperature*Pressure*Formaldehyde*StirringRate, data=Example6.7)
```

```
model2 <- lm(Filtration ~
↪  Temperature*Pressure*Formaldehyde*StirringRate + I(Temperature^2)
↪  + I(Pressure^2) + I(Formaldehyde^2) + I(StirringRate^2),
↪  data=Example6.7)
anova(model1, model2)
# Analysis of Variance Table
#
# Model 1: Filtration ~ Temperature * Pressure * Formaldehyde *
↪  StirringRate
# Model 2: Filtration ~ Temperature * Pressure * Formaldehyde *
↪  StirringRate +
#     I(Temperature^2) + I(Pressure^2) + I(Formaldehyde^2) +
↪  I(StirringRate^2)
#   Res.Df    RSS Df Sum of Sq      F Pr(>F)
# 1      4 50.263
# 2      3 48.750  1    1.5125 0.0931 0.7802
```

Above, the partial F-test shows that a quadratic model doesn't add much to the quality of our model.

Next, we examine if quadratic effects help a reduced model:

```
model1 <- lm(Filtration ~ Temperature + Formaldehyde + StirringRate +
↪  Temperature:Formaldehyde + Temperature:StirringRate,
↪  data=Example6.7)
model2 <- lm(Filtration ~ Temperature + Formaldehyde + StirringRate +
↪  Temperature:Formaldehyde + Temperature:StirringRate +
↪  I(Temperature^2) + I(Formaldehyde^2) + I(StirringRate^2),
↪  data=Example6.7)
anova(model1, model2)
# Analysis of Variance Table
#
# Model 1: Filtration ~ Temperature + Formaldehyde + StirringRate +
↪  Temperature:Formaldehyde +
#     Temperature:StirringRate
# Model 2: Filtration ~ Temperature + Formaldehyde + StirringRate +
↪  Temperature:Formaldehyde +
#     Temperature:StirringRate + I(Temperature^2) + I(Formaldehyde^2)
↪  +
#     I(StirringRate^2)
#   Res.Df    RSS Df Sum of Sq      F Pr(>F)
# 1     14 245.39
# 2     13 243.88  1    1.5125 0.0806 0.7809
```

Section 6.9 — Why We Work with Coded Design Variables

Section 6.9 compares the regression model on the natural units to the coded units. We begin with the natural units:

```
model <- lm(V ~ I * R, data=Table6.25)
summary(model)
#
# Call:
# lm.default(formula = V ~ I * R, data = Table6.25)
#
# Residuals:
#       1       2       3       4       5       6       7       8
# -0.1055  0.1055  0.0365 -0.0365 -0.1125  0.1125 -0.1365  0.1365
#
# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept)  -0.8055     0.8432  -0.955 0.393518
# I             0.1435     0.1654   0.868 0.434467
# R             0.4710     0.5333   0.883 0.427003
# I:R           0.9170     0.1046   8.768 0.000933 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 0.1479 on 4 degrees of freedom
# Multiple R-squared:  0.9988,  Adjusted R-squared:  0.9979
# F-statistic:  1086 on 3 and 4 DF,  p-value: 2.818e-06
```

In coded units this yields:

```
model <- lm(V ~ x1 * x2, data=Table6.25)
summary(model)
#
# Call:
# lm.default(formula = V ~ x1 * x2, data = Table6.25)
#
# Residuals:
#       1       2       3       4       5       6       7       8
# -0.1055  0.1055  0.0365 -0.0365 -0.1125  0.1125 -0.1365  0.1365
#
# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept)  7.49600    0.05229 143.349 1.42e-08 ***
# x1           1.51900    0.05229  29.049 8.36e-06 ***
# x2           2.52800    0.05229  48.344 1.10e-06 ***
# x1:x2        0.45850    0.05229   8.768 0.000933 ***
# ---
```

```
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 0.1479 on 4 degrees of freedom
# Multiple R-squared:  0.9988,  Adjusted R-squared:  0.9979
# F-statistic:  1086 on 3 and 4 DF,  p-value: 2.818e-06
```

The coded unit version produced a model where all of the terms were significant, whereas the natural-units model only found the interaction significant. Note that both models have the same R^2 values and so are equally good fits. But note how the standard error on the coefficients changes between the two formulations of the models.

References

Groemping, Ulrike, Boyko Amarov, and Hongquan Xu. 2019. *DoE.base: Full Factorials, Orthogonal Arrays and Base Utilities for Doe Packages*. https://cran.r-project.org/package=DoE.base.

Chapter 7

Blocking and Confounding in the Two-To-The-k Factorial Design

Chapter 7 of the text delves into handling blocking and confounding in the 2^k factorial designs. This is often necessary because, as the number of factors increases, it becomes exponentially more difficult to ensure homogeneous conditions.

Example 7.1

The example uses the data from Figure 6.1 of Section 6.2. Recall that the experimenter seeks to optimize yield by varying reactant concentration (`A`) and catalyst amount (`B`), but this time we'll run the experiment in three blocks as shown in Table 7.1. First, we model the blocks as fixed effects:

```
df <- MontgomeryDAE::Figure6.1
df[,'Block'] <- factor(c(rep(1, 4), rep(2, 4), rep(3, 4)))

model <- aov(Yield ~ Error(Block) + A * B, data=df)
summary(model)
#
# Error: Block
#           Df Sum Sq Mean Sq F value Pr(>F)
# Residuals  2    6.5    3.25
#
# Error: Within
#           Df Sum Sq Mean Sq F value   Pr(>F)
# A          1 208.33  208.33  50.336 0.000394 ***
```

```
# B          1  75.00   75.00  18.121 0.005340 **
# A:B        1   8.33    8.33   2.013 0.205710
# Residuals  6  24.83    4.14
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Next, we model the blocks as random effects:

```
library(lme4)
library(lmerTest)
model <- lmer(
    Yield ~ A * B + (1|Block),
    data=df,
    REML=TRUE
)
summary(model)
# Linear mixed model fit by REML. t-tests use Satterthwaite's method [
# lmerModLmerTest]
# Formula: Yield ~ A * B + (1 | Block)
#    Data: df
#
# REML criterion at convergence: 43.6
#
# Scaled residuals:
#     Min      1Q  Median      3Q     Max
# -1.0106 -0.6737 -0.2526  0.5474  1.5159
#
# Random effects:
#  Groups   Name        Variance Std.Dev.
#  Block    (Intercept) 0.000    0.000
#  Residual             3.917    1.979
# Number of obs: 12, groups:  Block, 3
#
# Fixed effects:
#             Estimate Std. Error      df t value Pr(>|t|)
# (Intercept)  27.5000     0.5713  8.0000  48.135 3.84e-11 ***
# A             4.1667     0.5713  8.0000   7.293 8.44e-05 ***
# B            -2.5000     0.5713  8.0000  -4.376  0.00236 **
# A:B           0.8333     0.5713  8.0000   1.459  0.18278
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Correlation of Fixed Effects:
#     (Intr) A     B
# A   0.000
# B   0.000  0.000
# A:B 0.000  0.000 0.000
# optimizer (nloptwrap) convergence code: 0 (OK)
# boundary (singular) fit: see ?isSingular
```

The blocking failed to show significance in the likelihood ratio test, shown next:

```
rand(model)
# ANOVA-like table for random-effects: Single term deletions
#
# Model:
# Yield ~ A + B + (1 | Block) + A:B
#             npar  logLik    AIC        LRT Df Pr(>Chisq)
# <none>         6 -21.782 55.565
# (1 | Block)    5 -21.782 53.565 1.4211e-14  1          1
```

Finally, we present confidence intervals for the main effects and for the variance components:

```
confint(model, oldNames=FALSE)
#                              2.5 %    97.5 %
# sd_(Intercept)|Block  0.0000000  2.055783
# sigma                 1.1351296  2.563095
# (Intercept)          26.5074479 28.492552
# A                     3.1741146  5.159219
# B                    -3.4925521 -1.507448
# A:B                  -0.1592188  1.825885
```

Example 7.2

We revisit the experiment from Example 6.2 where we are changing temperature, pressure, concentration of formaldehyde, and stirring rate to determine the effect on the filtration rate. Now we're going to use the highest order interaction term as a blocking scheme. First, we run the full model looking for the largest effects:

```
df <- Table6.10
df[df$Block == 1, 'Filtration'] <- df[df$Block == 1, 'Filtration'] -
↪  20

model <- aov(Filtration ~ Temperature * Pressure * Formaldehyde *
↪  StirringRate, data=Table6.10)
model.ss <- anova(model)[['Sum Sq']]
effect.estimates <- data.frame(
    'EffectEstimate'=2 * coef(model)[-1],
    'SumOfSquares'=model.ss[-1],
    'Contribution'=sprintf('%0.2f%%',
    ↪  100*model.ss[-1]/sum(model.ss[-1]))
)
```

	EffectEstimate	SumOfSquares	Contribution
Temperature	21.625	39.0625	1.01%
Pressure	3.125	390.0625	10.10%
Formaldehyde	9.875	855.5625	22.16%
StirringRate	14.625	0.0625	0.00%
Temperature:Pressure	0.125	1314.0625	34.04%
Temperature:Formaldehyde	-18.125	22.5625	0.58%
Pressure:Formaldehyde	2.375	1105.5625	28.64%
Temperature:StirringRate	16.625	0.5625	0.01%
Pressure:StirringRate	-0.375	5.0625	0.13%
Formaldehyde:StirringRate	-1.125	14.0625	0.36%
Temperature:Pressure:Formaldehyde	1.875	68.0625	1.76%
Temperature:Pressure:StirringRate	4.125	10.5625	0.27%
Temperature:Formaldehyde:StirringRate	-1.625	27.5625	0.71%
Pressure:Formaldehyde:StirringRate	-2.625	7.5625	0.20%
Temperature:Pressure:Formaldehyde:StirringRate	1.375	0.0000	0.00%

Next, we choose the largest effects and proceed with that reduced model while marking our `Block` as an error term:

```
model <- aov(Filtration ~ Error(Block) + Temperature + Formaldehyde +
↪  StirringRate + Temperature:Formaldehyde +
↪  Temperature:StirringRate, data=df)
summary(model)
#
# Error: Block
#           Df Sum Sq Mean Sq F value Pr(>F)
# Residuals  1   1388    1388
#
# Error: Within
#                           Df Sum Sq Mean Sq F value   Pr(>F)
# Temperature                1 1870.6  1870.6   89.76 5.60e-06 ***
# Formaldehyde               1  390.1   390.1   18.72 0.001915 **
# StirringRate               1  855.6   855.6   41.05 0.000124 ***
# Temperature:Formaldehyde   1 1314.1  1314.1   63.05 2.35e-05 ***
# Temperature:StirringRate   1 1105.6  1105.6   53.05 4.65e-05 ***
# Residuals                  9  187.6    20.8
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Example 7.3

We return to the plasma etching experiment of Example 6.1. This time we're adding blocking! Note that I've added a `Block` for convenience, but in the R output we see it noticed we were aliasing `Gap:Flow` and `Gap:Flow:Power` interactions with blocks:

```
model <- aov(EtchRate ~ Error(Block) + Gap * Flow * Power,
↪  data=Table6.4)
summary(model)
#
# Error: Block
#                Df Sum Sq Mean Sq
# Power           1 374850  374850
# Gap:Flow        1   2475    2475
# Gap:Flow:Power  1    127     127
#
# Error: Within
#            Df Sum Sq Mean Sq F value   Pr(>F)
# Gap         1  41311   41311  18.339 0.002679 **
# Flow        1    218     218   0.097 0.763911
# Gap:Power   1  94403   94403  41.909 0.000193 ***
# Flow:Power  1     18      18   0.008 0.930849
# Residuals   8  18021    2253
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Chapter 8

Two-Level Fractional Factorial Designs

Construction of full factorial designs is rather trivial, but construction of fractions of two-level designs can be tricky. The package `FrF2` (Groemping 2019) has methods to create and visualize both regular fractions (where full confounding occurs between effects) and non-regular fractions (where partial aliasing occurs). The function `FrF2` is very flexible in how you can request a design. Below, I have requested a four-factor resolution III design:

```
library(FrF2)

FrF2(nfactors=4, resolution=3)
#    A  B  C  D
# 1 -1  1 -1  1
# 2  1  1 -1 -1
# 3  1 -1 -1  1
# 4  1 -1  1 -1
# 5 -1 -1 -1 -1
# 6  1  1  1  1
# 7 -1  1  1 -1
# 8 -1 -1  1  1
# class=design, type= FrF2
```

Next, I request a design on five factors using eight runs:

```
FrF2(nfactors=5, nruns=8)
#    A  B  C  D  E
# 1 -1  1 -1 -1  1
# 2 -1  1  1 -1 -1
# 3  1 -1 -1 -1 -1
# 4 -1 -1  1  1 -1
# 5  1  1 -1  1 -1
# 6 -1 -1 -1  1  1
```

```
# 7  1  1  1  1  1
# 8  1 -1  1 -1  1
# class=design, type= FrF2
```

Example 8.1

We return again to the filtration rate experiment, except this time we'll run it as a 2^{4-1}_{IV} design with `D` aliased with the `A:B:C` interaction (see Table 8.3):

```
model <-lm(FiltrationRate ~ A + B + C + D + A:B + A:C + A:D,
↪  data=Table8.3)
print(summary(model))
#
# Call:
# lm.default(formula = FiltrationRate ~ A + B + C + D + A:B + A:C +
#     A:D, data = Table8.3)
#
# Residuals:
# ALL 8 residuals are 0: no residual degrees of freedom!
#
# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept)    70.75         NA      NA       NA
# A               9.50         NA      NA       NA
# B               0.75         NA      NA       NA
# C               7.00         NA      NA       NA
# D               8.25         NA      NA       NA
# A:B            -0.50         NA      NA       NA
# A:C            -9.25         NA      NA       NA
# A:D             9.50         NA      NA       NA
#
# Residual standard error: NaN on 0 degrees of freedom
# Multiple R-squared:      1,   Adjusted R-squared:    NaN
# F-statistic:   NaN on 7 and 0 DF,  p-value: NA
```

Since effects A, C, and D are large we remove B and the A:B interaction:

```
model <-lm(FiltrationRate ~ A + C + D + A:C + A:D, data=Table8.3)
print(summary(model))
#
# Call:
# lm.default(formula = FiltrationRate ~ A + C + D + A:C + A:D,
#     data = Table8.3)
#
# Residuals:
```

```
#     1     2     3     4     5     6     7     8
# -1.25 -0.25  1.25  0.25 -1.25 -0.25  1.25  0.25
#
# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept)  70.7500     0.6374  111.00 8.11e-05 ***
# A             9.5000     0.6374   14.90  0.00447 **
# C             7.0000     0.6374   10.98  0.00819 **
# D             8.2500     0.6374   12.94  0.00592 **
# A:C          -9.2500     0.6374  -14.51  0.00471 **
# A:D           9.5000     0.6374   14.90  0.00447 **
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 1.803 on 2 degrees of freedom
# Multiple R-squared:  0.9979,  Adjusted R-squared:  0.9926
# F-statistic: 188.6 on 5 and 2 DF,  p-value: 0.005282
```

Example 8.2

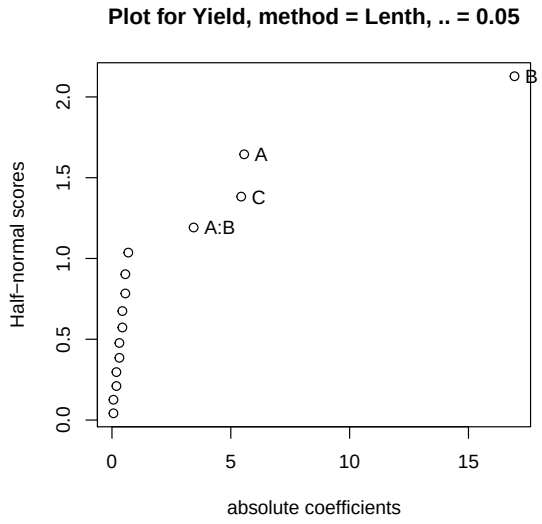
An integrated circuit manufacturing process was studied to understand how aperture setting (`A`), exposure time (`B`), develop time (`C`), and mask dimension (`D`) affected yield. The design used was a 2^{5-1} design with `E` aliased to the `A:B:C:D` interaction:

```
model <- aov(Yield ~ . * ., data=Table8.5)
model.ss <- anova(model)[['Sum Sq']]
effect.estimates <- data.frame(
    'EffectEstimate'=2 * coef(model)[-1],
    'SumOfSquares'=model.ss[-1],
    'Contribution'=sprintf('%0.2f%%',
    ↪  100*model.ss[-1]/sum(model.ss[-1]))
)
```

	EffectEstimate	SumOfSquares	Contribution
A	11.125	4590.0625	86.93%
B	33.875	473.0625	8.96%
C	10.875	3.0625	0.06%
D	-0.875	1.5625	0.03%
E	0.625	189.0625	3.58%
A:B	6.875	0.5625	0.01%
A:C	0.375	5.0625	0.10%
A:D	1.125	5.0625	0.10%
A:E	1.125	1.5625	0.03%
B:C	0.625	0.0625	0.00%
B:D	-0.125	0.0625	0.00%
B:E	-0.125	3.0625	0.06%
C:D	0.875	0.5625	0.01%
C:E	0.375	7.5625	0.14%
D:E	-1.375	0.0000	0.00%

The terms `A`, `B`, `C` and `A:B` have large magnitude, but we'll verify that they look like active effects with a half-normal probability plot:

```
library(DoE.base)
vals <- halfnormal(model)
```



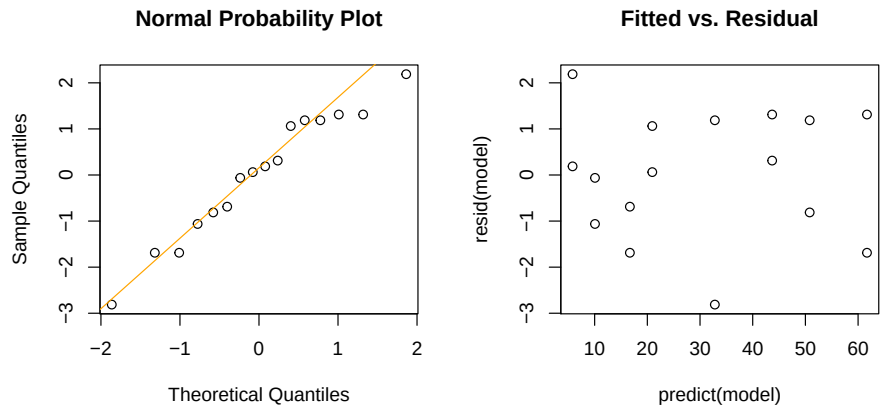
```
vals$signif
# [1] "A:B" "C"   "A"   "B"
```

Next, we fit the reduced model:

```
model <- aov(Yield ~ A + B + C + A:B, data=Table8.5)
print(summary(model))
#             Df Sum Sq Mean Sq F value   Pr(>F)
# A            1    495     495  193.19 2.53e-08 ***
# B            1   4590    4590 1791.24 1.56e-13 ***
# C            1    473     473  184.61 3.21e-08 ***
# A:B          1    189     189   73.78 3.30e-06 ***
# Residuals   11     28       3
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Checking the diagnostic plots, it looks like there's no strong evidence our assumptions have been violated:

```
op <- par(mfrow=c(1,2))
qqnorm(resid(model), main='Normal Probability Plot')
qqline(resid(model), col='orange')
plot(x=predict(model), y=resid(model), main='Fitted vs. Residual')
```




```
par(op)
```

Example 8.4

A common defect that can occur in injection molding is excessive shrinkage. An investigation into how mold temperature (A), screw speed (B), holding time

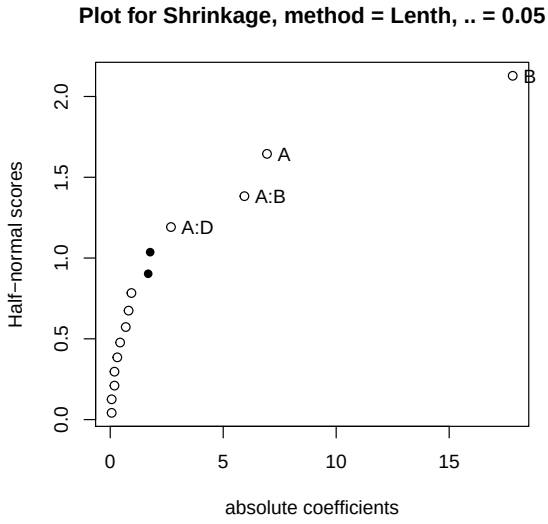
(C), cycle time (D), gate size (E), and holding pressure (F) affect shrinkage so that shrinkage can be better controlled. A 2_{IV}^{6-2} design was used with E aliased to ABC and F aliased to B:C:D:

```
model <- lm(Shrinkage ~ . * ., data=Table8.10)
print(summary(model))
#
# Call:
# lm.default(formula = Shrinkage ~ . * ., data = Table8.10)
#
# Residuals:
#      1      2      3      4      5      6      7      8      9
↪  10
#  2.375 -2.375  2.500 -2.500 -2.500  2.500 -2.375  2.375 -2.375
↪  2.375
#     11     12     13     14     15     16
# -2.500  2.500  2.500 -2.500  2.375 -2.375
#
# Coefficients: (8 not defined because of singularities)
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept)  27.3125     1.7241  15.841  0.00396 **
# A             6.9375     1.7241   4.024  0.05657 .
# B            17.8125     1.7241  10.331  0.00924 **
# C            -0.4375     1.7241  -0.254  0.82339
# D             0.6875     1.7241   0.399  0.72862
# E             0.1875     1.7241   0.109  0.92333
# F             0.1875     1.7241   0.109  0.92333
# A:B           5.9375     1.7241   3.444  0.07496 .
# A:C          -0.8125     1.7241  -0.471  0.68387
# A:D          -2.6875     1.7241  -1.559  0.25939
# A:E          -0.9375     1.7241  -0.544  0.64112
# A:F           0.3125     1.7241   0.181  0.87288
# B:C               NA         NA      NA       NA
# B:D          -0.0625     1.7241  -0.036  0.97438
# B:E               NA         NA      NA       NA
# B:F          -0.0625     1.7241  -0.036  0.97438
# C:D               NA         NA      NA       NA
# C:E               NA         NA      NA       NA
# C:F               NA         NA      NA       NA
# D:E               NA         NA      NA       NA
# D:F               NA         NA      NA       NA
# E:F               NA         NA      NA       NA
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 6.897 on 2 degrees of freedom
# Multiple R-squared:  0.9857,  Adjusted R-squared:  0.8929
# F-statistic: 10.62 on 13 and 2 DF,  p-value: 0.08928
```

The only main effects that are active are A and B. The A:B interaction is large,

but it is aliased with `C:E`. Since neither `C` nor `E` is active we'll assume it's the `A:B` interaction that's active. Compare this to a half-normal probability plot:

```
halfnormal(model)
#
# The following effects are completely aliased:
# [1] B:C B:E C:D C:E C:F D:E D:F E:F
#
# Significant effects (alpha=0.05, Lenth method):
# [1] B   A   A:B A:D
```



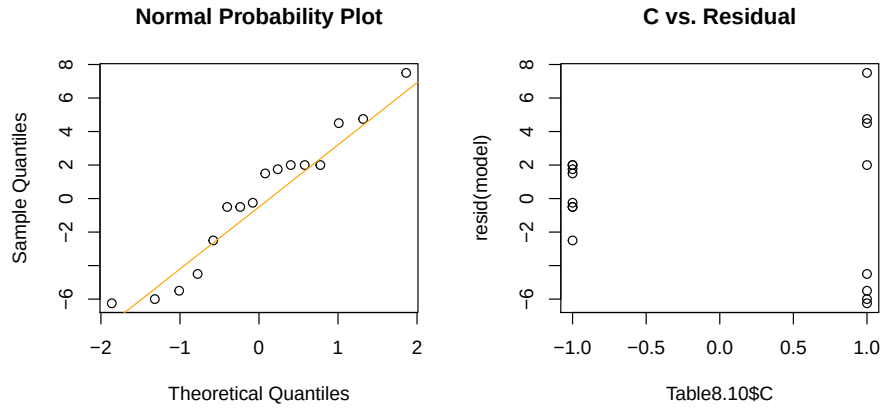
Because of this we decide to keep `A`, `B`, and the `A:B` interaction:

```
model <- lm(Shrinkage ~ A + B + A:B, data=Table8.10)
print(summary(model))
#
# Call:
# lm.default(formula = Shrinkage ~ A + B + A:B, data = Table8.10)
#
# Residuals:
#    Min     1Q Median     3Q    Max
# -6.250 -3.000  0.625  2.000  7.500
#
# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept)   27.312      1.138  23.996 1.65e-11 ***
# A              6.938      1.138   6.095 5.38e-05 ***
# B             17.812      1.138  15.649 2.39e-09 ***
# A:B            5.938      1.138   5.216 0.000216 ***
# ---
```

```
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 4.553 on 12 degrees of freedom
# Multiple R-squared:  0.9626,  Adjusted R-squared:  0.9533
# F-statistic: 103.1 on 3 and 12 DF,  p-value: 7.837e-09
```

Next, let's check the normal probability plot and the residual versus holding time:

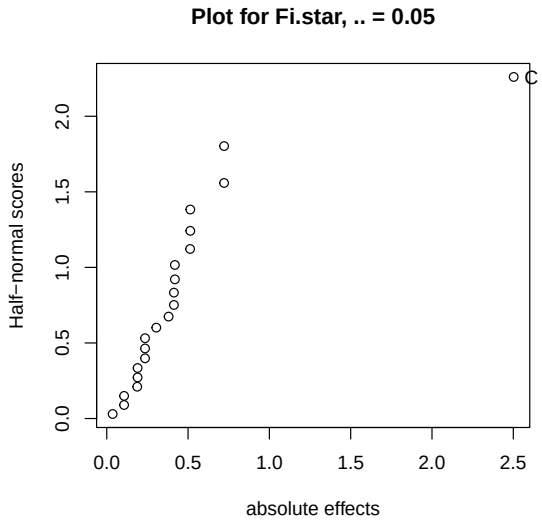
```
op <- par(mfrow=c(1,2))
qqnorm(resid(model), main='Normal Probability Plot')
qqline(resid(model), col='orange')
plot(x=Table8.10$C, y=resid(model), main='C vs. Residual')
```




```
par(op)
```

It looks like the variance is much larger at the longer holding time (`C`) than at the smaller holding time. Next, we model the dispersion:

```
ix <- Table8.10$C == 1
X <- model.matrix(Shrinkage ~ -1 + . * ., data=Table8.10)
Fi.star <- apply(
    X,
    2,
    function(w){
        log(
            sd(resid(model)[w == 1])^2
            /
            sd(resid(model)[w != 1])^2
        )
    }
)
vals <- halfnormal(Fi.star)
```




```
vals
# [1] "C"
```

The plot suggests that we can model dispersion using only `C`.

Example 8.6

A set of eight factors were selected to model the deviation from profile of the output of a CNC machine. The factors are:

- `A` - x-Axis shift
- `B` - y-Axis shift
- `C` - z-Axis shift
- `D` - Tool supplier
- `E` - a-Axis shift
- `F` - Spindle speed
- `G` - Fixture height
- `H` - Feed rate

In addition, we are to block on spindle, so we choose to use the `E:H` interaction and the `A:B:E` interaction chain to represent the four blocks. Our analysis of the 2^{8-3}_{IV} design in Table 8.16 proceeds as follows:

```
model <- aov(log(Deviation) ~ Error(Block) + (A + B + C + D + E + F +
↪  G + H)^3, data=Table8.16)
print(summary(model))
#
# Error: Block
#       Df   Sum Sq  Mean Sq
# E:H    1 0.009993 0.009993
# A:B:E  1 0.008874 0.008874
# A:B:H  1 0.001273 0.001273
#
# Error: Within
#       Df Sum Sq Mean Sq
# A      1 0.6740  0.6740
# B      1 0.3217  0.3217
# C      1 0.0053  0.0053
# D      1 0.0935  0.0935
# E      1 0.0000  0.0000
# F      1 0.0120  0.0120
# G      1 0.1078  0.1078
# H      1 0.0016  0.0016
# A:B    1 0.0003  0.0003
# A:C    1 0.0308  0.0308
# A:D    1 1.1197  1.1197
# A:E    1 0.0005  0.0005
# A:F    1 0.0162  0.0162
# A:G    1 0.0224  0.0224
# A:H    1 0.0203  0.0203
# B:E    1 0.0776  0.0776
# B:H    1 0.0014  0.0014
# C:D    1 0.0095  0.0095
# C:E    1 0.0127  0.0127
# C:G    1 0.0017  0.0017
# C:H    1 0.0296  0.0296
# D:E    1 0.0023  0.0023
# D:H    1 0.0020  0.0020
# E:F    1 0.0026  0.0026
# E:G    1 0.0231  0.0231
# F:H    1 0.0063  0.0063
# G:H    1 0.0002  0.0002
# A:C:D  1 0.0236  0.0236
```

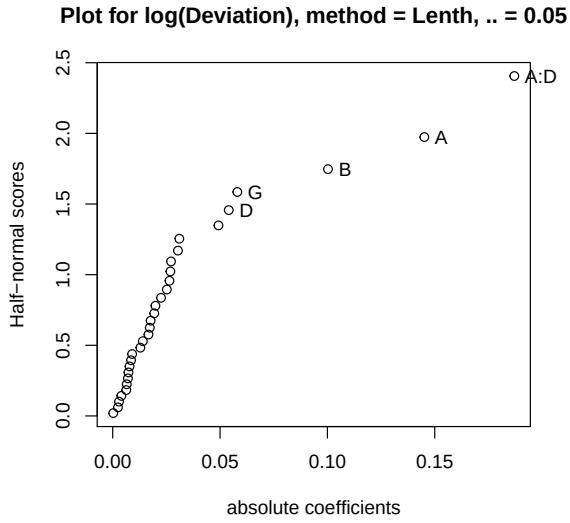
```
model.ss <- summary(model$Within)[[1]][['Sum Sq']]
coef.est <- coef(model$Within)
effect.estimates <- data.frame(
    'EffectEstimate'=2 * coef.est,
    'SumOfSquares'=model.ss,
    'Contribution'=sprintf('%0.2f%%', 100*model.ss/sum(model.ss))
)
```

	EffectEstimate	SumOfSquares	Contribution
A	0.2902628	0.6740199	25.74%
B	-0.2005396	0.3217292	12.29%
C	-0.0257642	0.0053104	0.20%
D	0.1081317	0.0935397	3.57%
E	-0.0005063	0.0000021	0.00%
F	-0.0387104	0.0119880	0.46%
G	0.1160816	0.1077994	4.12%
H	0.0141688	0.0016060	0.06%
A:B	-0.0058808	0.0002767	0.01%
A:C	-0.0620632	0.0308147	1.18%
A:D	-0.3741164	1.1197046	42.76%
A:E	0.0080389	0.0005170	0.02%
A:F	-0.0450200	0.0162144	0.62%
A:G	0.0528791	0.0223696	0.85%
A:H	-0.0504222	0.0203392	0.78%
B:E	0.0985058	0.0776271	2.96%
B:H	0.0130888	0.0013705	0.05%
C:D	0.0345228	0.0095346	0.36%
C:E	0.0398201	0.0126851	0.48%
C:G	-0.0146680	0.0017212	0.07%
C:H	0.0607952	0.0295685	1.13%
D:E	0.0170813	0.0023342	0.09%
D:H	0.0156873	0.0019687	0.08%
E:F	-0.0180815	0.0026155	0.10%
E:G	-0.0537096	0.0230778	0.88%
F:H	-0.0280798	0.0063078	0.24%
G:H	0.0048912	0.0001914	0.01%
A:C:D	-0.0543339	0.0236174	0.90%

Process knowledge helps us to decide `A`, `B`, `D`, and `A:D` should be included in the analysis. The `halfnormal` plot does not like `aov` models with error terms, so let's remodel it and just be mindful of what factors represent blocks:

```
model <- aov(log(Deviation) ~ (A + B + C + D + E + F + G + H)^3,
↪  data=Table8.16)
vals <- halfnormal(model)
#
# The following effects are completely aliased:
#  [1] B:C   B:D   B:F   B:G   C:F   D:F   D:G   F:G   A:B:C A:B:D
↪  A:B:F
```

```
# [12] A:B:G A:C:E A:C:F A:C:G A:C:H A:D:E A:D:F A:D:G A:D:H A:E:F
↪  A:E:G
# [23] A:E:H A:F:G A:F:H A:G:H B:C:D B:C:E B:C:F B:C:G B:C:H B:D:E
↪  B:D:F
# [34] B:D:G B:D:H B:E:F B:E:G B:E:H B:F:G B:F:H B:G:H C:D:E C:D:F
↪  C:D:G
# [45] C:D:H C:E:F C:E:G C:E:H C:F:G C:F:H C:G:H D:E:F D:E:G D:E:H
↪  D:F:G
# [56] D:F:H D:G:H E:F:G E:F:H E:G:H F:G:H
#
# Significant effects (alpha=0.05, Lenth method):
# [1] A:D A   B   G   D
```




```
vals$signif
# [1] "D"   "G"   "B"   "A"   "A:D"
```

The plot seems to confirm that `A`, `B`, `D`, and `A:D` are probably active. We confirm this with the following ANOVA table:

```
model <- aov(log(Deviation) ~ Error(Block) + A + B + D + A:D,
↪  data=Table8.16)
print(summary(model))
#
# Error: Block
#           Df  Sum Sq  Mean Sq F value Pr(>F)
# Residuals  3 0.02014 0.006713
#
# Error: Within
#           Df Sum Sq Mean Sq F value   Pr(>F)
```

```
# A          1 0.6740  0.6740  39.469 1.71e-06 ***
# B          1 0.3217  0.3217  18.839 0.000222 ***
# D          1 0.0935  0.0935   5.477 0.027908 *
# A:D        1 1.1197  1.1197  65.567 2.55e-08 ***
# Residuals 24 0.4099  0.0171
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Example 8.7

An experiment to understand how eye focus time is affected by acuity or sharpness of vision (`A`), distance from target to eye (`B`), target shape (`C`), illumination level (`D`), target size (`E`), target density (`F`), and subject (`G`) is in Table 8.21. The design is a 2^{7-4}_{III} fractional factorial. First, let's select the candidate active main effects:

```
model <- lm(Time ~ ., data=Table8.21)
print(2*coef(model)[-1])
#      A      B      C      D      E      F      G
# 20.625 38.375 -0.275 28.875 -0.275 -0.625 -2.425
```

As described in the text, when we project down to just `A`, `B`, and `D` we don't get a full factorial in those main effects, but instead we get a replicated 2^{3-1}. To disentangle the aliases we need to do a foldover as described in Table 8.22:

```
df <- rbind(Table8.21, Table8.22)
model <- lm(Time ~ (A + B + D)^2, data=df)
summary(model)
#
# Call:
# lm.default(formula = Time ~ (A + B + D)^2, data = df)
#
# Residuals:
#    Min     1Q Median     3Q    Max
# -4.575 -1.025  0.125  1.925  2.825
#
# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept)  98.6375     0.7067 139.582 2.53e-16 ***
# A             0.7375     0.7067   1.044    0.324
# B            19.0250     0.7067  26.922 6.52e-10 ***
# D            14.6875     0.7067  20.784 6.47e-09 ***
```

```
# A:B          -0.2500     0.7067  -0.354    0.732
# A:D           0.1625     0.7067   0.230    0.823
# B:D           9.5750     0.7067  13.550 2.72e-07 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 2.827 on 9 degrees of freedom
# Multiple R-squared:  0.9933,  Adjusted R-squared:  0.9889
# F-statistic: 223.6 on 6 and 9 DF,  p-value: 2.845e-09
```

Example 8.8

Example 8.8 uses a non-regular fraction called a Plackett-Burman design. These designs have partial aliasing, and we can fit them with model selection procedures such as stepwise regression.

The text analyzes this with stepwise regression based on p-values requiring a p-value less than 0.1 to enter and a p-value greater than 0.25 to leave. R's `MASS` package provides stepwise selection based on AIC-like criteria, but does not support p-values directly. While some other R packages do support p-value based stepwise regression they fail to support strong or weak heredity as found in JMP and in `stepAIC` in `MASS`.

For convenience, this package has included `p.stepwise` which does stepwise regression under strong heredity assumptions, e.g. `X1:X2` can only enter the model if `X1` and `X2` are in the model, and `X1` can only leave the model if no interaction involving it is in the model. The results of this model fitting procedure are given below:

```
alpha.to.enter <- 0.10
alpha.to.leave <- 0.10
model <- lm(y~1, data=Table8.25)
scope <- y ~ (X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9 + X10 + X11 +
↪  X12)^2

final.model <- p.stepwise(model, scope, alpha.to.enter,
↪  alpha.to.leave)
# Beginning Stepwise Regression
#   Model: y ~ 1
# Steps:
#   Adding X2 at p=0.0218. Model sigma=21.012160, R^2=25.9%,
↪  Adj-R^2=21.8%
```

```
#   Adding X4 at p=0.0368. Model sigma=18.948837, R^2=43.1%,
↪  Adj-R^2=36.4%
#   Adding X1 at p=0.0560. Model sigma=17.363746, R^2=55.1%,
↪  Adj-R^2=46.6%
#   Adding X1:X2 at p=0.0004. Model sigma=11.716646, R^2=80.8%,
↪  Adj-R^2=75.7%
#   Adding X1:X4 at p=0.0000. Model sigma=5.999990, R^2=95.3%,
↪  Adj-R^2=93.6%
#   Adding X5 at p=0.0622. Model sigma=5.419264, R^2=96.4%,
↪  Adj-R^2=94.8%
summary(final.model)
#
# Call:
# lm.default(formula = y ~ X2 + X4 + X1 + X5 + X2:X1 + X4:X1, data =
↪  Table8.25)
#
# Residuals:
#      Min       1Q   Median       3Q      Max
# -11.1862  -0.7644   1.1167   3.0220   6.2152
#
# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept)  200.000      1.212 165.046  < 2e-16 ***
# X2             9.892      1.238   7.991 2.26e-06 ***
# X4            12.108      1.238   9.781 2.33e-07 ***
# X1             8.000      1.212   6.602 1.71e-05 ***
# X5             2.582      1.266   2.040   0.0622 .
# X2:X1        -12.538      1.265  -9.915 1.99e-07 ***
# X4:X1          9.538      1.265   7.542 4.24e-06 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 5.419 on 13 degrees of freedom
# Multiple R-squared:  0.9644,  Adjusted R-squared:  0.948
# F-statistic: 58.74 on 6 and 13 DF,  p-value: 1.145e-08
```

Section 8.7.2

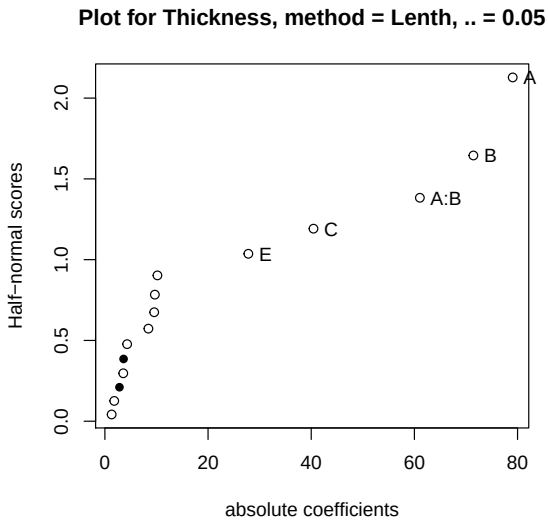
Section 8.7.2. analyzes an experiment where six factors were analyzed to determine their effect on the thickness of a coating on wafer. The factors were as follows:

- Spin speed (`A`)
- Acceleration (`B`)
- Volume of resist applied (`C`)
- Spin time (`D`)

- Resist viscosity (`E`)
- Exhaust rate (`F`)

We begin with the half-normal probability plot for the data in Table 8.31:

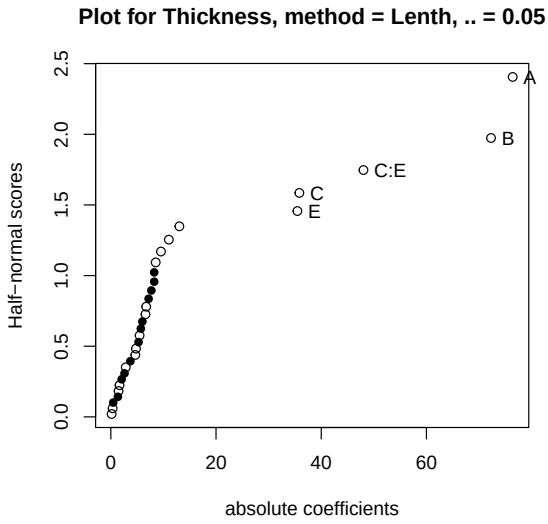
```
model <- lm(Thickness ~ . * ., data=Table8.31)
vals <- halfnormal(model)
```




```
vals$signif
# [1] "E"   "C"   "A:B" "B"   "A"
```

This suggests that either `A:B` or `C:E` is large because the two are aliased in our original design. To address this we do a complete fold-over on `A`, as given in Table 8.32:

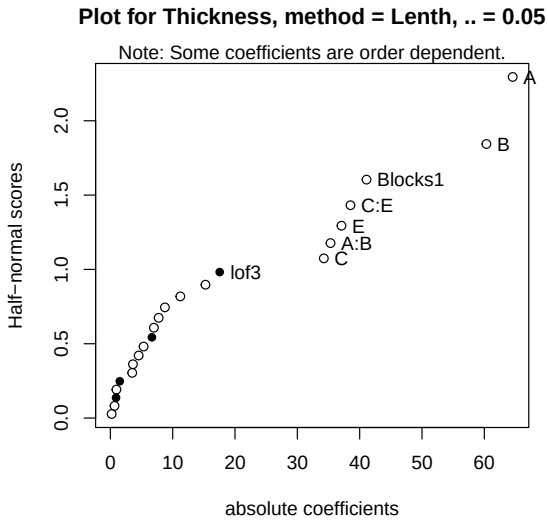
```
model <- lm(Thickness ~ Blocks + (A + B + C + D + E + F)^2,
↪  data=Table8.32)
vals <- halfnormal(model)
```




```
vals$signif
# [1] "E"   "C"   "C:E" "B"   "A"
```

Another option would have been to do a partial fold-over selecting the runs where A is at its low level. This design is given in Table 8.33, but is the same as Table 8.32 where block 2 has A at the low level:

```
ix <- Table8.32[,'Blocks'] == 1 | (Table8.32[,'Blocks'] == 2 &
↪  Table8.32[,'A'] == -1)
Table8.33 <- Table8.32[ix,]
model <- lm(Thickness ~ Blocks + (A + B + C + D + E + F)^2,
↪  data=Table8.33)
vals <- halfnormal(model, ME.partial=TRUE)
```




```
vals$signif
# [1] "lof3"    "C"       "A:B"     "E"       "C:E"     "Blocks1" "B"
↪
# [8] "A"
```

Note that `A:B` is now in the model, as is a term for the second block. We may ignore this second term as it was necessary to include Block as a main effect to use the `halfnormal` function.

```
model <- aov(Thickness ~ Error(Blocks) + (A + B + C + D + E + F)^2,
↪  data=Table8.33)
print(summary(model))
#
# Error: Blocks
#   Df Sum Sq Mean Sq
# A  1  40542   40542
#
# Error: Within
#           Df Sum Sq Mean Sq F value  Pr(>F)
# A          1 100014  100014  46.923 0.00238 **
# B          1  87363   87363  40.987 0.00306 **
# C          1  28154   28154  13.209 0.02207 *
# D          1     11      11   0.005 0.94700
# E          1  33004   33004  15.484 0.01703 *
# F          1   1838    1838   0.862 0.40571
# A:B        1  29950   29950  14.051 0.01997 *
# A:C        1   3024    3024   1.419 0.29944
# A:D        1    295     295   0.138 0.72874
# A:E        1      1       1   0.000 0.98359
# A:F        1   1170    1170   0.549 0.49986
```

```
# B:C         1   5588    5588   2.621 0.18074
# B:D         1    683     683   0.320 0.60168
# B:E         1    315     315   0.148 0.72020
# B:F         1   1441    1441   0.676 0.45706
# C:E         1  35627   35627  16.715 0.01500 *
# D:E         1    495     495   0.232 0.65503
# E:F         1     23      23   0.011 0.92301
# Residuals   4   8526    2131
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Though the half-normal result differs from the text, the ANOVA table supports our decision to include `A:B` in the model.

References

Groemping, Ulrike. 2019. *FrF2: Fractional Factorial Designs with 2-Level Factors*. https://cran.r-project.org/package=FrF2.

Chapter 9

Additional Designs and Analysis Topics for Factorial and Fractional Factorial Designs

Chapter 9 in the text introduces designs with three-level factors and some non-regular fractional factorial (two-level) designs. The chapter also introduces optimal designs which (as of this writing) are not very well supported in R.

Example 9.1

The syrup loss experiment of Table 9.1 investigated how nozzle design, the filling speed, and the operating pressure affect the amount of syrup lost while filling containers. Each factor is investigated at three levels using a 3^3 design. Note that we want to treat `Speed` and `Pressure` as factors for the ANOVA model:

```
model <- aov(SyrupLoss ~ NozzleType * factor(Speed) *
↪  factor(Pressure), data=Table9.1)
print(summary(model))
#                                          Df Sum Sq Mean Sq F value
# NozzleType                                2    994     497   1.165
# factor(Speed)                             2  61190   30595  71.735
# factor(Pressure)                          2  69105   34553  81.014
# NozzleType:factor(Speed)                  4   6301    1575   3.693
# NozzleType:factor(Pressure)               4   7514    1878   4.404
# factor(Speed):factor(Pressure)            4  12854    3214   7.535
```

```
# NozzleType:factor(Speed):factor(Pressure)  8   4629     579   1.357
# Residuals                                 27  11516     426
#                                             Pr(>F)
# NozzleType                                0.327102
# factor(Speed)                             1.57e-11 ***
# factor(Pressure)                          3.89e-12 ***
# NozzleType:factor(Speed)                  0.015950 *
# NozzleType:factor(Pressure)               0.007187 **
# factor(Speed):factor(Pressure)            0.000327 ***
# NozzleType:factor(Speed):factor(Pressure) 0.259496
# Residuals
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

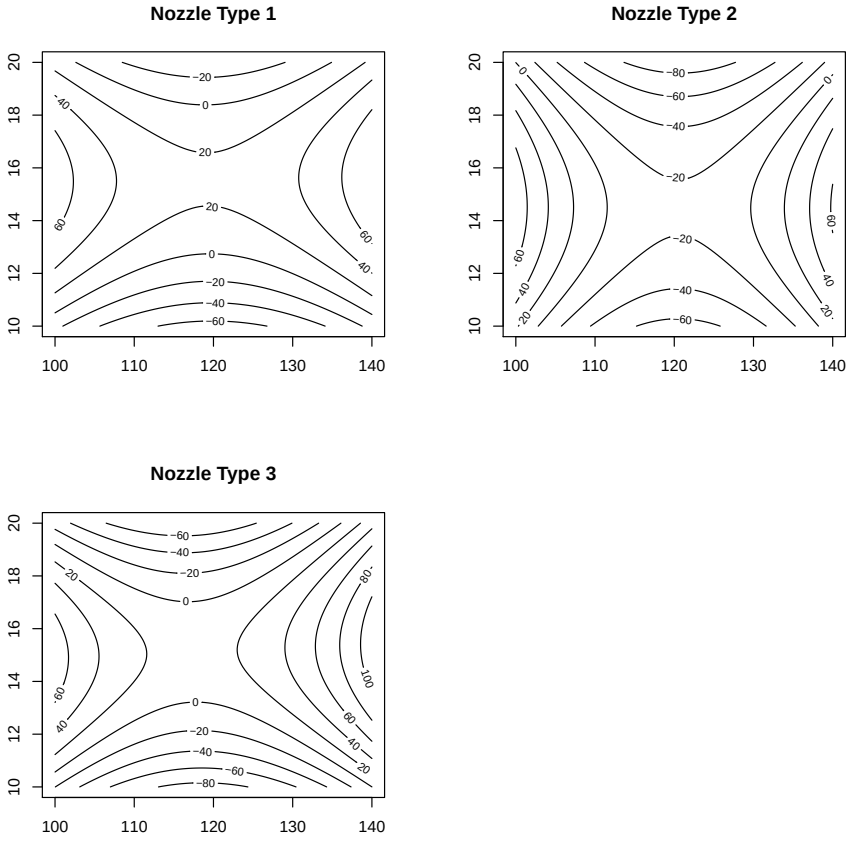
Next we'll fit regression models for each nozzle type to fit a contour plot for each nozzle type:

```
code.speed <- function(x){
    (x - 120) / (140 - 120)
}

code.pressure <- function(x){
    (x - 15) / (20 - 15)
}

df.reg <- data.frame(
    'NozzleType'=Table9.1$NozzleType,
    'Speed'=code.speed(Table9.1$Speed),
    'Pressure'=code.pressure(Table9.1$Pressure),
    'SyrupLoss'=Table9.1$SyrupLoss
)

model1 <- lm(SyrupLoss ~ Speed * Pressure + I(Speed^2) +
↪  I(Pressure^2), data=df.reg[df.reg$NozzleType == 1,])
model2 <- lm(SyrupLoss ~ Speed * Pressure + I(Speed^2) +
↪  I(Pressure^2), data=df.reg[df.reg$NozzleType == 2,])
model3 <- lm(SyrupLoss ~ Speed * Pressure + I(Speed^2) +
↪  I(Pressure^2), data=df.reg[df.reg$NozzleType == 3,])

print(coef(model1))
#    (Intercept)          Speed       Pressure     I(Speed^2)
↪  I(Pressure^2)
#       22.05556        3.50000       16.33333       51.66667
↪  -71.83333
# Speed:Pressure
#        2.87500
print(coef(model2))
#    (Intercept)          Speed       Pressure     I(Speed^2)
↪  I(Pressure^2)
#     -17.833333      -5.083333     -12.250000      84.250000
↪  -60.250000
```

```
# Speed:Pressure
#      -0.750000
print(coef(model3))
#    (Intercept)           Speed       Pressure     I(Speed^2)
↪  I(Pressure^2)
#      15.111111      20.333333       5.916667      75.833333
↪  -94.916667
# Speed:Pressure
#      10.500000
```

Next, we generate contour plots with the following code:

```
x <- seq(from=100, to=140, length.out=100)
y <- seq(from=10, to=20, length.out=100)
df0 <- expand.grid('Speed'=code.speed(x), 'Pressure'=code.pressure(y))

z1 <- matrix(predict(model1, df0), ncol=100)
z2 <- matrix(predict(model2, df0), ncol=100)
z3 <- matrix(predict(model3, df0), ncol=100)

par(mfrow=c(2,2))
contour(x, y, z1, main='Nozzle Type 1')
contour(x, y, z2, main='Nozzle Type 2')
contour(x, y, z3, main='Nozzle Type 3')
plot.new()  # This just uses up the last cell.
```



Example 9.2

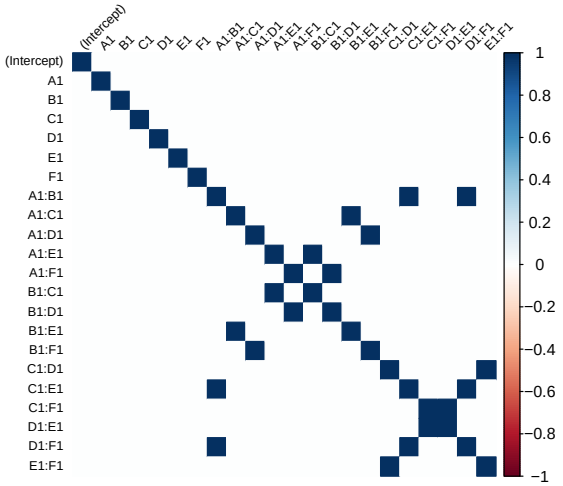
This example illustrates the design of a 3^2 in three blocks (aliasing the AB^2 effect with blocks):

```
model <- aov(Response ~ Error(Block) + A + B + A:B, data=Example9.2)
print(summary(model))
#
# Error: Block
#     Df Sum Sq Mean Sq
# A:B  2  10.89   5.444
#
# Error: Within
#     Df Sum Sq Mean Sq
# A    2 131.56   65.78
# B    2   0.22    0.11
# A:B  2   2.89    1.44
```

Figure 9.10

Constructing correlation plots is easiest to do using the `corrplot` package (Wei et al. 2017). Here, I construct the correlation plot for the 2^{6-2} Resolution IV design of Figure 9.10:

```
library(FrF2)
# Loading required package: DoE.base
# Loading required package: grid
# Loading required package: conf.design
# Registered S3 method overwritten by 'DoE.base':
#   method           from
#   factorize.factor conf.design
#
# Attaching package: 'DoE.base'
# The following objects are masked from 'package:stats':
#
#     aov, lm
# The following object is masked from 'package:graphics':
#
#     plot.design
# The following object is masked from 'package:base':
#
#     lengths
library(corrplot)
# corrplot 0.84 loaded

X.d <- FrF2(nfactors=6, nruns=16)
X.m <- model.matrix(~.*., data=X.d)
X.corr <- t(X.m) %*% X.m / 16
corrplot(X.corr, method='color', tl.col="black", tl.srt=45,
↪  tl.cex=0.7)
```



Example 9.3 — The Spin Coating Experiment

Recall the spin coating experiment from Section 8.7.2. Again we use stepwise regression.

```
model <- p.stepwise(
    lm(Thickness ~ 1, Table9.24),
    ~ (A + B + C + D + E + F)^2,
    alpha.to.enter=0.20,
    alpha.to.leave=0.20
)
# Beginning Stepwise Regression
#   Model: Thickness ~ 1
# Steps:
#   Adding A at p=0.0001. Model sigma=86.672413, R^2=66.0%,
↪  Adj-R^2=63.5%
#   Adding B at p=0.0134. Model sigma=70.474900, R^2=79.1%,
↪  Adj-R^2=75.9%
#   Adding C at p=0.0466. Model sigma=61.777996, R^2=85.2%,
↪  Adj-R^2=81.5%
#   Adding E at p=0.1735. Model sigma=59.085647, R^2=87.6%,
↪  Adj-R^2=83.1%
#   Adding C:E at p=0.0022. Model sigma=37.975979, R^2=95.3%,
↪  Adj-R^2=93.0%
#   Adding B:E at p=0.1133. Model sigma=34.555511, R^2=96.5%,
↪  Adj-R^2=94.2%
print(summary(model))
#
# Call:
# lm.default(formula = Thickness ~ A + B + C + E + C:E + B:E, data =
↪  Table9.24)
#
# Residuals:
#     Min      1Q  Median      3Q     Max
# -63.875  -5.375   3.625  16.625  45.125
#
# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept) 4462.875      8.639 516.604  < 2e-16 ***
# A             85.500     10.580   8.081 2.04e-05 ***
# B            -77.750     10.580  -7.348 4.33e-05 ***
# C            -25.500      9.975  -2.556  0.03087 *
# E             21.500      8.639   2.489  0.03449 *
# C:E           54.750     12.217   4.481  0.00153 **
# B:E          -17.500      9.975  -1.754  0.11327
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
```

```
# Residual standard error: 34.56 on 9 degrees of freedom
# Multiple R-squared:  0.9652,  Adjusted R-squared:  0.942
# F-statistic: 41.63 on 6 and 9 DF,  p-value: 4.602e-06
print(anova(model))
# Analysis of Variance Table
#
# Response: Thickness
#           Df Sum Sq Mean Sq  F value    Pr(>F)
# A          1 203852  203852 170.7186 3.718e-07 ***
# B          1  40602   40602  34.0029 0.0002495 ***
# C          1  18769   18769  15.7183 0.0032812 **
# E          1   7396    7396   6.1939 0.0344927 *
# C:E        1  23980   23980  20.0828 0.0015294 **
# B:E        1   3675    3675   3.0777 0.1132691
# Residuals  9  10747    1194
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Example 9.4

At present there's no easy way to construct I-optimal designs with blocks using R. We will construct an I-optimal design for six factors at levels $(-1, 0, 1)$. The design from the text has balanced blocks with three runs in each block, but the `AlgDesign` package does not support this constraint for I-optimal designs. We construct the I-optimal design (without blocking) as follows:

```
library(AlgDesign)

candidate.list <- expand.grid(
    'A'=c(-1, 0, 1),
    'B'=c(-1, 0, 1),
    'C'=c(-1, 0, 1),
    'D'=c(-1, 0, 1),
    'E'=c(-1, 0, 1),
    'F'=c(-1, 0, 1)
)

opt.i <- optFederov(~A+B+C+D+E+F, data=candidate.list, nTrials=12,
↪  criterion='I', approximate=FALSE, nRepeats=1000)
```

The optimal design discovered has I-criterion 5 and the following layout:

	A	B	C	D	E	F
19	-1	-1	1	-1	-1	-1
63	1	1	-1	1	-1	-1
79	-1	1	1	1	-1	-1
171	1	1	-1	-1	1	-1
183	1	-1	1	-1	1	-1
217	-1	-1	-1	1	1	-1
487	-1	-1	-1	-1	-1	1
513	1	1	1	-1	-1	1
543	1	-1	-1	1	-1	1
655	-1	1	-1	-1	1	1
723	1	-1	1	1	1	1
727	-1	1	1	1	1	1

Compare this to the design in Table 9.36. The value of the I-criterion for this design is as follows:

```
eval.design(~ A + B + C + D + E + F, design=Table9.36,
↪  X=candidate.list)$I
# [1] 6.027472
```

References

Wei, Taiyun, Viliam Simko, Michael Levy, Yihui Xie, Yan Jin, and Jeff Zemela. 2017. *Corrplot: Visualization of a Correlation Matrix*. https://cran.r-project.org/package=corrplot.

Chapter 10

Fitting Regression Models

Chapter 10 of the text discusses fitting regression models. Thankfully, R's support for such models is very good. Again, we must be careful to set the contrasts of unordered factors to "contr.sum" as follows:

```
options(contrasts=c(unordered='contr.sum', ordered='contr.poly'))
```

Example 10.1

The example investigates the relationship of reaction temperature and catalyst feed rate to the viscosity of a polymer. First, let's do the regression through matrix operations. Define the X matrix using the `model.matrix` command:

```
X <- model.matrix(~ReactionTemperature + CatalystFeedRate,
↪  data=Table10.2)
print(X)
#    (Intercept) ReactionTemperature CatalystFeedRate
# 1            1                  80                8
# 2            1                  93                9
# 3            1                 100               10
# 4            1                  82               12
# 5            1                  90               11
# 6            1                  99                8
# 7            1                  81                8
# 8            1                  96               10
# 9            1                  94               12
# 10           1                  93               11
# 11           1                  97               13
# 12           1                  95               11
# 13           1                 100                8
# 14           1                  85               12
# 15           1                  86                9
# 16           1                  87               12
```

```
# attr(,"assign")
# [1] 0 1 2
```

The response y is just the `Viscosity` column:

```
y <- Table10.2$Viscosity
print(y)
#  [1] 2256 2340 2426 2293 2330 2368 2250 2409 2364 2379 2440 2364
↪  2404 2317
# [15] 2309 2328
```

Now we solve $(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}y$. We can find the inverse $(X^{\mathrm{T}}X)^{-1}$ with `solve(t(X) %*% X)` and then multiply by $X^{\mathrm{T}}y$, but it's numerically more stable to supply $X^{\mathrm{T}}y$ as the second argument to `solve`:

```
beta.hat <- solve(t(X) %*% X, t(X) %*% y)
print(beta.hat)
#                             [,1]
# (Intercept)         1566.077771
# ReactionTemperature    7.621290
# CatalystFeedRate       8.584846
```

Now, to do this using `lm` is far simpler:

```
model <- lm(Viscosity ~ ReactionTemperature + CatalystFeedRate,
↪  data=Table10.2)
print(summary(model))
#
# Call:
# lm(formula = Viscosity ~ ReactionTemperature + CatalystFeedRate,
#     data = Table10.2)
#
# Residuals:
#      Min       1Q   Median       3Q      Max
# -21.4972 -13.1978  -0.4736  10.5558  25.4299
#
# Coefficients:
#                     Estimate Std. Error t value Pr(>|t|)
# (Intercept)         1566.0778    61.5918   25.43 1.80e-12 ***
# ReactionTemperature    7.6213     0.6184   12.32 1.52e-08 ***
# CatalystFeedRate       8.5848     2.4387    3.52  0.00376 **
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 16.36 on 13 degrees of freedom
# Multiple R-squared:  0.927,   Adjusted R-squared:  0.9157
# F-statistic:  82.5 on 2 and 13 DF,  p-value: 4.1e-08
```

Table 10.3 gives many types of residuals and diagnostic measures, many of which we'll create below:

```
Table10.3 <- data.frame(
    'y'=Table10.2$Viscosity,
    'Predicted'=predict(model),
    'Residual'=resid(model),
    'hii'=hatvalues(model),
    'StudentizedResids'=rstudent(model),
    'CooksD'=cooks.distance(model)
)
```

y	Predicted	Residual	hii	StudentizedResids	CooksD
2256	2244.460	11.5402553	0.3495069	0.8662386	0.1370211
2340	2352.121	-12.1213615	0.1024725	-0.7697800	0.0232810
2426	2414.055	11.9447620	0.1766710	0.7931568	0.0463190
2293	2294.042	-1.0417084	0.2510838	-0.0707119	0.0006051
2330	2346.427	-16.4271831	0.0768901	-1.0492249	0.0303303
2368	2389.264	-21.2642561	0.2653280	-1.6060332	0.2768744
2250	2252.081	-2.0810347	0.3193512	-0.1482818	0.0037185
2409	2383.570	25.4299223	0.0979706	1.7648944	0.0969908
2364	2385.497	-21.4971893	0.1418941	-1.4825374	0.1109258
2379	2369.291	9.7089467	0.0798914	0.6034150	0.0110803
2440	2416.946	23.0540946	0.2783574	1.7952141	0.3538676
2364	2384.534	-20.5336335	0.0961841	-1.3632128	0.0618388
2404	2396.886	7.1144538	0.2894812	0.5008633	0.0361525
2317	2316.906	0.0944214	0.1851984	0.0061435	0.0000031
2309	2298.772	10.2276690	0.1341527	0.6570582	0.0233161
2328	2332.148	-4.1481587	0.1555667	-0.2659017	0.0046761

Example 10.6

Consider the viscosity data from Table 10.2. If we wish to test $H_0 : \beta_2 = 0$ versus $H_1 : \beta_2 \neq 0$ by partial F-test we can use the `anova` function:

```
model01 <- lm(Viscosity ~ ReactionTemperature, data=Table10.2)
model012 <- lm(Viscosity ~ ReactionTemperature + CatalystFeedRate,
↪  data=Table10.2)
print(anova(model01, model012))
```

```
# Analysis of Variance Table
#
# Model 1: Viscosity ~ ReactionTemperature
# Model 2: Viscosity ~ ReactionTemperature + CatalystFeedRate
#   Res.Df    RSS Df Sum of Sq      F   Pr(>F)
# 1     14 6795.1
# 2     13 3478.9  1    3316.2 12.392 0.003765 **
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

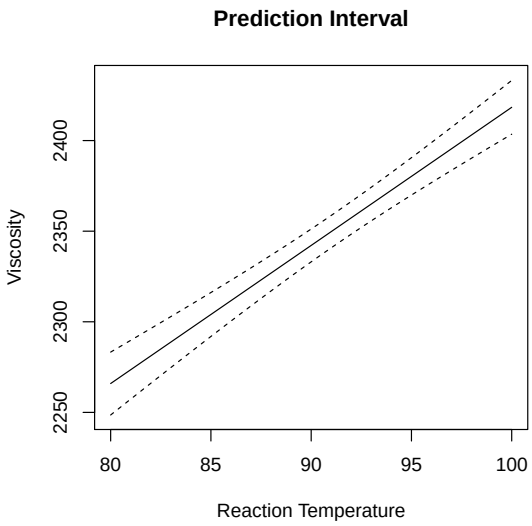
Example 10.7

Parameter confidence intervals can be retrieved with the `confint` function:

```
model <- lm(Viscosity ~ ReactionTemperature + CatalystFeedRate,
↪  data=Table10.2)
confint(model)
#                            2.5 %      97.5 %
# (Intercept)          1433.016700 1699.138843
# ReactionTemperature     6.285254    8.957326
# CatalystFeedRate        3.316390   13.853302
```

We can also get other intervals. For example, the prediction interval for the response over `ReactionTemperature` at `CatalystFeedRate` set to 10.5 is given below:

```
x1 <- seq(from=80, to=100, length.out=100)
x2 <- rep(10.5, 100)
y <-predict(model, newdata=data.frame('ReactionTemperature'=x1,
↪  'CatalystFeedRate'=x2), interval="confidence")
plot(x1, y[,1], type='l', ylim=c(2248, 2434), main='Prediction
↪  Interval', xlab='Reaction Temperature', ylab='Viscosity')
lines(x1, y[,2], lty=2)
lines(x1, y[,3], lty=2)
```


Prediction Interval
Viscosity
2250
2300
2350
2400
80
85
90
95
100
Reaction Temperature

Chapter 11

Response Surface Methods and Designs

A key feature of many response surface designs is splitting the residual error into a pure error term and a lack of fit term. The pure error term is invariant to the specified model as it compares the mean of runs at a specific location with the observed values. This allows testing model misspecification, but isn't supported by native R, so we will use the `rsm` package (Lenth 2018) in this chapter. The `rsm` package also has facilities for design construction — the vignette for the package is a good resource on the many features.

Example 11.1

An engineer investigating the effect of reaction time and reaction temperature on yield collected the data of Table 11.1. We'll use `rsm`'s `coded.data` function to encode our data on the $[-1, 1]$ scale, and then fit a first order model using the `FO` helper in the formula. Note that we can use `code2val` to convert coded values to natural units.

```
library(rsm)

df <- Table11.1
cf <- coded.data(df, x1 ~ (Time - 35) / 5, x2 ~ (Temperature - 155) /
↪  5)
model <- rsm(Yield ~ FO(x1, x2), data=cf)
summary(model)
#
# Call:
# rsm(formula = Yield ~ FO(x1, x2), data = cf)
```

```
#
#              Estimate Std. Error  t value  Pr(>|t|)
# (Intercept) 40.444444   0.057288 705.9869 5.451e-16 ***
# x1           0.775000   0.085932   9.0188  0.000104 ***
# x2           0.325000   0.085932   3.7821  0.009158 **
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Multiple R-squared:  0.941,   Adjusted R-squared:  0.9213
# F-statistic: 47.82 on 2 and 6 DF,  p-value: 0.0002057
#
# Analysis of Variance Table
#
# Response: Yield
#             Df  Sum Sq Mean Sq F value    Pr(>F)
# FO(x1, x2)   2 2.82500 1.41250 47.8213 0.0002057
# Residuals    6 0.17722 0.02954
# Lack of fit  2 0.00522 0.00261  0.0607 0.9419341
# Pure error   4 0.17200 0.04300
#
# Direction of steepest ascent (at radius 1):
#        x1        x2
# 0.9221944 0.3867267
#
# Corresponding increment in original units:
#        Time Temperature
#    4.610972    1.933633
```

The lack of fit is not significant so there's no need to add terms, but let's add the two-factor interaction term into the model using the `TWI` helper anyway.

```
model <- rsm(Yield ~ FO(x1, x2) + TWI(x1, x2), data=cf)
summary(model)
#
# Call:
# rsm(formula = Yield ~ FO(x1, x2) + TWI(x1, x2), data = cf)
#
#              Estimate Std. Error  t value  Pr(>|t|)
# (Intercept) 40.444444   0.062311 649.0693 1.648e-13 ***
# x1           0.775000   0.093467   8.2917 0.0004166 ***
# x2           0.325000   0.093467   3.4772 0.0177127 *
# x1:x2       -0.025000   0.093467  -0.2675 0.7997870
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Multiple R-squared:  0.9418,  Adjusted R-squared:  0.9069
# F-statistic: 26.97 on 3 and 5 DF,  p-value: 0.00163
#
# Analysis of Variance Table
#
```

```
# Response: Yield
#             Df  Sum Sq Mean Sq F value    Pr(>F)
# FO(x1, x2)   2 2.82500 1.41250 40.4213 0.0008188
# TWI(x1, x2)  1 0.00250 0.00250  0.0715 0.7997870
# Residuals    5 0.17472 0.03494
# Lack of fit  1 0.00272 0.00272  0.0633 0.8137408
# Pure error   4 0.17200 0.04300
#
# Stationary point of response surface:
# x1 x2
# 13 31
#
# Stationary point in original units:
#        Time Temperature
#         100         310
#
# Eigenanalysis:
# eigen() decomposition
# $values
# [1]  0.0125 -0.0125
#
# $vectors
#          [,1]       [,2]
# x1 -0.7071068 -0.7071068
# x2  0.7071068 -0.7071068
```

We could fit a second order model using the `SO` helper, but there is no present need.

Example 11.2

We conduct the test for lack of fit over the linear, linear with interaction, and the full quadratic model, for each of the responses. First, we begin with `Yield`. For `Yield`, the first order model shows lack of fit:

```
cf <- coded.data(Table11.6, x1 ~ (Time - 85) / 5, x2 ~ (Temperature -
↪  175) / 5)
model <- rsm(Yield ~ FO(x1, x2), data=cf)
summary(model)$lof
# Analysis of Variance Table
#
# Response: Yield
#             Df Sum Sq Mean Sq F value    Pr(>F)
# FO(x1, x2)   2 10.043  5.0215  2.6853 0.1165617
# Residuals   10 18.700  1.8700
# Lack of fit  6 18.488  3.0814 58.1387 0.0007595 ***
```

```
# Pure error   4  0.212  0.0530
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The first order model with interactions shows lack of fit as well:

```
model <- rsm(Yield ~ FO(x1, x2) + TWI(x1, x2), data=cf)
summary(model)$lof
# Analysis of Variance Table
#
# Response: Yield
#             Df Sum Sq Mean Sq F value   Pr(>F)
# FO(x1, x2)   2 10.043  5.0215  2.4495 0.141472
# TWI(x1, x2)  1  0.250  0.2500  0.1220 0.734960
# Residuals    9 18.450  2.0500
# Lack of fit  5 18.238  3.6476 68.8231 0.000571 ***
# Pure error   4  0.212  0.0530
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The second order model does not show significant lack of fit:

```
model <- rsm(Yield ~ SO(x1, x2), data=cf)
summary(model)$lof
# Analysis of Variance Table
#
# Response: Yield
#             Df  Sum Sq Mean Sq  F value    Pr(>F)
# FO(x1, x2)   2 10.0430  5.0215  70.8143 2.267e-05 ***
# TWI(x1, x2)  1  0.2500  0.2500   3.5256    0.1025
# PQ(x1, x2)   2 17.9537  8.9769 126.5944 3.194e-06 ***
# Residuals    7  0.4964  0.0709
# Lack of fit  3  0.2844  0.0948   1.7885    0.2886
# Pure error   4  0.2120  0.0530
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Next, we examine `Viscosity`. Lack of fit is no longer significant at the second order model:

```
model.fo <- rsm(Viscosity ~ FO(x1, x2), data=cf)
model.in <- rsm(Viscosity ~ FO(x1, x2) + TWI(x1, x2), data=cf)
model.so <- rsm(Viscosity ~ SO(x1, x2), data=cf)

rbind(
    'First Order LOF'=summary(model.fo)$lof[3,],
    'First Order Interaction LOF'=summary(model.in)$lof[4,],
```

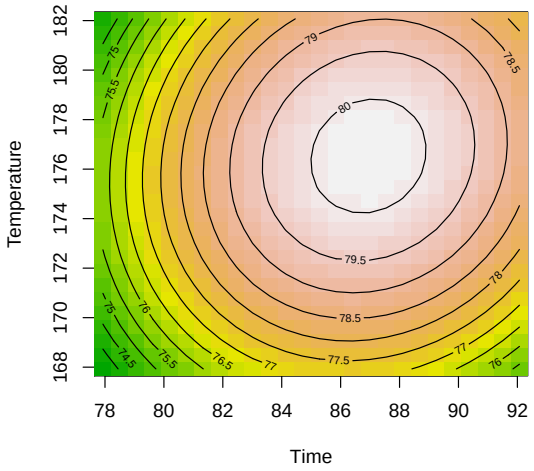
```
    'Second Order LOF'=summary(model.so)$lof[5,]
)
# Analysis of Variance Table
#
# Response: Viscosity
#                              Df Sum Sq Mean Sq F value   Pr(>F)
# First Order LOF               6 343.84  57.307 22.9229 0.004613 **
# First Order Interaction LOF   5 337.59  67.519 27.0075 0.003517 **
# Second Order LOF              3  26.22   8.741  3.4963 0.128988
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Finally, we examine `MolecularWeight`. Lack of fit is not significant for the first order model, so this is the model we shall use:

```
model.fo <- rsm(MolecularWeight ~ FO(x1, x2), data=cf)
model.in <- rsm(MolecularWeight ~ FO(x1, x2) + TWI(x1, x2), data=cf)
model.so <- rsm(MolecularWeight ~ SO(x1, x2), data=cf)

rbind(
    'First Order LOF'=summary(model.fo)$lof[3,],
    'First Order Interaction LOF'=summary(model.in)$lof[4,],
    'Second Order LOF'=summary(model.so)$lof[5,]
)
# Analysis of Variance Table
#
# Response: MolecularWeight
#                              Df Sum Sq Mean Sq F value Pr(>F)
# First Order LOF               6 208591   34765  2.1160 0.2443
# First Order Interaction LOF   5 182991   36598  2.2275 0.2289
# Second Order LOF              3 142150   47383  2.8839 0.1663
```

Constructing the response surface plots is pretty easy with `rsm`. Figure 11.11 gives the response surface for the second order model of `Yield`, which we present below:

```
contour(model, ~ x1 + x2, image = TRUE)
```

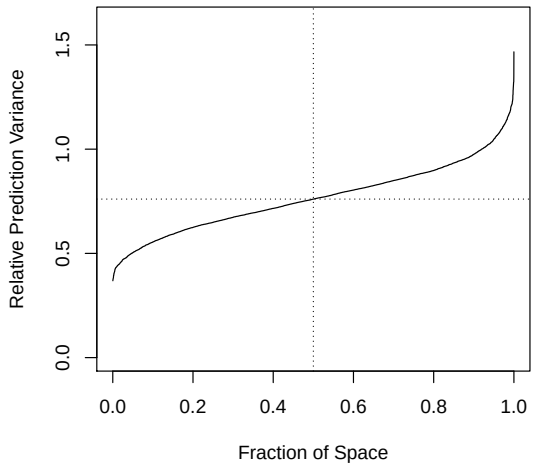


Example 11.3

Note that the `AlgDesign` package uses the Federov exchange algorithm and chooses from a candidate set of points, whereas software like JMP and Design Expert use an efficient search algorithm that does not require enumeration of points. As of this time I'm unaware of an R package for constructing such designs, however the existing optimization routines in R are sufficient for such construction.

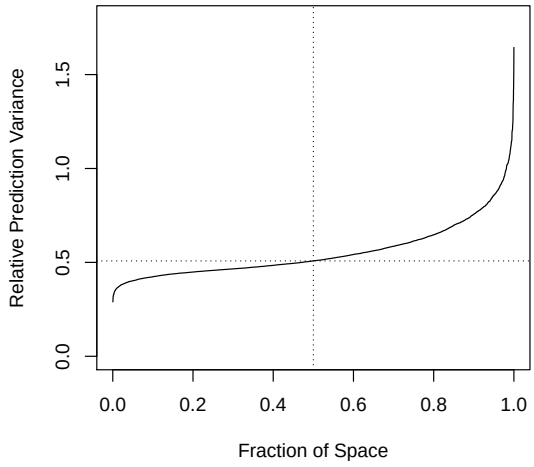
For both the D-optimal design in Table 11.13 and the I-optimal design in Table 11.14, a fraction of design space plot was created. The package `Vdgraph` (Lawson 2014) provides the ability to construct such plots. The following is the fraction of design space plot for the D-optimal design of Table 11.13:

```
library(Vdgraph)

FDSPlot(Table11.13[,c('X1','X2','X3','X4')], mod=2)
```



Next, we present the fraction of design space plot for the I-optimal design of Table 11.14:

```
FDSPlot(Table11.14[,c('X1','X2','X3','X4')], mod=2)
```

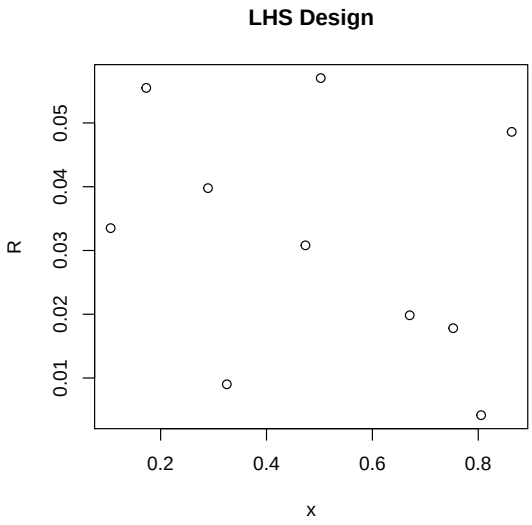


Example 11.4

Computational fluid dynamics (CFD) models of exhaust from a jet turbine was studied to determine the effect of location in the exhaust plume on temperature. The R packages `DiceDesign` (Helbert 2018) and `DiceEval` (Dupuy and Helbert 2015) are explicitly for the design and analysis of computer experiments (https://www.jstatsoft.org/article/view/v065i11/v65i11.pdf).

We can construct many space filling designs with DiceDesign, though the specific sphere-packing design from JMP is not supported in this package. Let's construct a Latin Hypercube design for the space $x \in (0.05, 0.095)$ and $R \in (0, 0.062)$:

```
library(DiceDesign)
df <- lhsDesign(10, 2)$design
colnames(df) <- c('x', 'R')
# Translate the points which are on [0,1] scale to the required scale
df[,'x'] <- df[,'x'] * 0.9 + 0.05
df[,'R'] <- df[,'R'] * 0.062
plot(df, main='LHS Design')
```



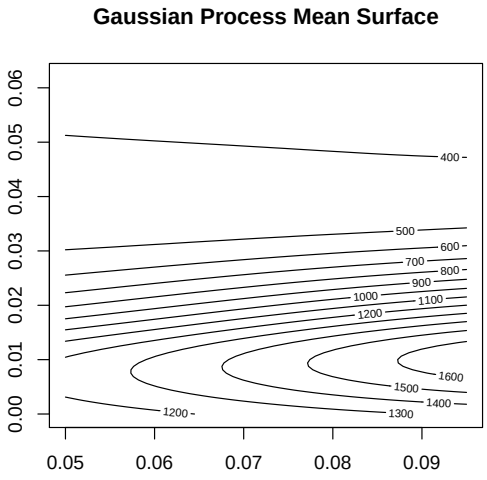
Let us now move to the design in Table 11.16. After fitting the model we will create a plot of the estimated mean response surface:

```
library(DiceEval)
# Loading required package: DiceKriging
model <- modelFit(Table11.16[,c('x','R')], Table11.16[,'Temperature'],
↪  type='Kriging')
#
# optimisation start
# ------------------
# * estimation method   : MLE
# * optimisation method : BFGS
# * analytical gradient : used
# * trend model : ~1
# * covariance model :
#   - type :  matern5_2
```

```
#   - nugget : NO
#   - parameters lower bounds :  1e-10 1e-10
#   - parameters upper bounds :  0.09 0.124
#   - best initial criterion value(s) :  -67.40055
#
# N = 2, M = 5 machine precision = 2.22045e-16
# At X0, 0 variables are exactly at the bounds
# At iterate     0  f=       67.401  |proj g|=     0.016832
# At iterate     1  f =       67.389  |proj g|=      0.016509
# At iterate     2  f =       67.298  |proj g|=       0.11154
# At iterate     3  f =       67.131  |proj g|=        0.1123
# At iterate     4  f =       67.021  |proj g|=       0.11123
# At iterate     5  f =       67.015  |proj g|=       0.11071
# At iterate     6  f =       67.015  |proj g|=      0.035788
# At iterate     7  f =       67.015  |proj g|=    6.8414e-05
#
# iterations 7
# function evaluations 9
# segments explored during Cauchy searches 8
# BFGS updates skipped 0
# active bounds at final generalized Cauchy point 1
# norm of the final projected gradient 6.84145e-05
# final function value 67.0153
#
# F = 67.0153
# final  value 67.015282
# converged
```

The contour plot of the response surface is given below. Compare this to Table 11.17's Contour Profiler result:

```
x.range <- range(Table11.16[,'x'])
R.range <- range(Table11.16[,'R'])
x <- seq(from=x.range[1], to=x.range[2], length.out=100)
R <- seq(from=R.range[1], to=R.range[2], length.out=100)
z <- matrix(
    modelPredict(model, expand.grid('x'=x, 'R'=R)),
    nrow=length(x)
)
contour(x, R, z, main='Gaussian Process Mean Surface')
```



Example 11.5 — A Three-Component Mixture

The yarn elongation experiment investigates how `Polyethylene`, `Polystyrene`, and `Polypropylene` affect yarn elongation in kilograms of force applied. A $\{3, 2\}$ simplex lattice design was used to investigate this mixture experiment. The package `mixexp` (Lawson, Willden, and Piepel 2016) provides a way to construct mixture experiments with R (https://www.jstatsoft.org/index.php/jss/article/view/v072c02/v72c02.pdf), and does the package `qualityTools` (Roth 2016) (https://CRAN.R-project.org/package=qualityTools). Analysis of mixture experiments can use the regular `lm` function:

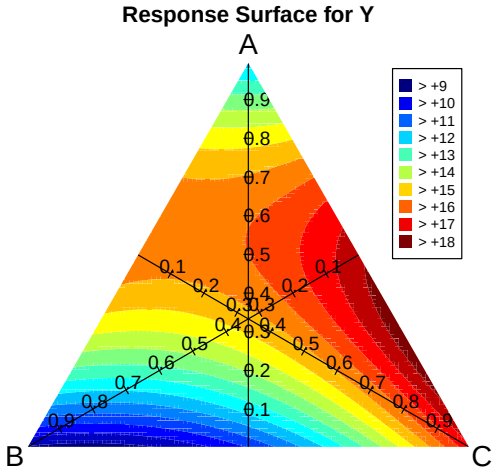
Note: `qualityTools` was recently removed from CRAN. You can install the last hosted version by downloading it manually from https://cran.r-project.org/src/contrib/Archive/qualityTools/qualityTools_1.55.tar.gz.

```
model <- lm(Elongation ~ -1 + (Polyethylene + Polystyrene +
↪  Polypropylene)^2, data=Table11.19)
print(summary(model))
#
# Call:
# lm(formula = Elongation ~ -1 + (Polyethylene + Polystyrene +
#     Polypropylene)^2, data = Table11.19)
#
# Residuals:
```

```
#    Min     1Q Median     3Q    Max
#  -0.80  -0.50  -0.30   0.65   1.30
#
# Coefficients:
#                            Estimate Std. Error t value Pr(>|t|)
# Polyethylene                11.7000     0.6037  19.381 1.20e-08 ***
# Polystyrene                  9.4000     0.6037  15.571 8.15e-08 ***
# Polypropylene               16.4000     0.6037  27.166 6.01e-10 ***
# Polyethylene:Polystyrene    19.0000     2.6082   7.285 4.64e-05 ***
# Polyethylene:Polypropylene  11.4000     2.6082   4.371  0.00180 **
# Polystyrene:Polypropylene   -9.6000     2.6082  -3.681  0.00507 **
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 0.8537 on 9 degrees of freedom
# Multiple R-squared:  0.9977,  Adjusted R-squared:  0.9962
# F-statistic: 658.1 on 6 and 9 DF,  p-value: 2.271e-11
```

Here, I use the `qualityTools` package to create a visualization of the mixture model:

```
library(qualityTools)

Y <- Table11.19$Elongation
mdo <- mixDesign(3, 2, center=FALSE, axial=FALSE, randomize=FALSE,
↪  replicates=c(1,1,2,3))
response(mdo) <- Y
# [1] "Y"
contourPlot3(A, B, C, Y, data=mdo, form='quadratic')
```



Example 11.6

Construction of a D-optimal constrained mixture experiment can be achieved using the `Xvert` function in the `mixexp` package to generate a host of candidate points, then using the `AlgDesign` package to select a D-optimal subset of these points. Analysis of the optimal experiment in Example 11.6 proceeds as follows.

We cannot use the `rsm` package because it has trouble with dropping the intercept. So instead we'll use a helper function in this package `pe.summary`. First, we need to code the units:

```
df <- Table11.20
for(s in c('Monomer', 'Crosslinker', 'Resin')){
    df[,s] <- df[,s] / 100
}
```

Next, we examine the hardness model:

```
library(mixexp)

model <- MixModel(df, "Hardness", c("Monomer", "Crosslinker",
↪  "Resin"), 2)
#
#                      coefficients   Std.err    t.value       Prob
# Monomer                319.23727 230.85842  1.3828270 0.20408780
# Crosslinker           -217.93997 252.99060 -0.8614548 0.41406105
# Resin                   45.11084  67.78145  0.6655337 0.52442970
# Crosslinker:Monomer   1114.68013 632.33615  1.7627968 0.11595184
# Resin:Monomer        -1097.89960 398.50308 -2.7550593 0.02486381
# Crosslinker:Resin      347.90891 582.86433  0.5968952 0.56708180
#
# Residual standard error:  4.859673  on  8 degrees of freedom
# Corrected Multiple R-squared:  0.5966083
```

Fitting the ANOVA table we find no lack of fit:

```
model <- aov(Hardness ~ -1 + (Monomer + Crosslinker + Resin)^2,
↪  data=df)
pe.summary(model)
#                          Df      Sum Sq     Mean Sq     F value
# Regression                6 8880.372296 1480.062049  47.3619856
#   Monomer                 1 6291.393456 6291.393456 201.3245906
#   Crosslinker             1 2181.508123 2181.508123  69.8082599
#   Resin                   1  156.886635  156.886635   5.0203723
#   Monomer:Crosslinker     1   64.860051   64.860051   2.0755216
#   Monomer:Resin           1  177.394255  177.394255   5.6766162
```

```
#   Crosslinker:Resin      1    8.329776    8.329776   0.2665528
# Error                    8  188.627704   23.578463          NA
#   Lack of Fit            8   63.627704    7.953463   0.2545108
#   Pure Error             4  125.000000   31.250000          NA
#                              Pr(>F)
# Regression               0.0011347431
#   Monomer                0.0001432556
#   Crosslinker            0.0011219163
#   Resin                  0.0885611530
#   Monomer:Crosslinker    0.2231112019
#   Monomer:Resin          0.0757775849
#   Crosslinker:Resin      0.6328828560
# Error                            NA
#   Lack of Fit            0.9527347346
#   Pure Error                       NA
```

Next, we examine the `Solids` model:

```
model <- MixModel(df, "Solids", c("Monomer", "Crosslinker", "Resin"),
↪  2)
#
#                      coefficients   Std.err   t.value         Prob
# Monomer                 507.4676 274.84874  1.846352 1.020462e-01
# Crosslinker            1467.1927 301.19823  4.871186 1.238108e-03
# Resin                   611.4802  80.69727  7.577458 6.439721e-05
# Crosslinker:Monomer   -1884.5790 752.82846 -2.503331 3.675078e-02
# Resin:Monomer         -1383.6080 474.43825 -2.916308 1.940069e-02
# Crosslinker:Resin     -3857.1446 693.92974 -5.558408 5.356675e-04
#
# Residual standard error:  5.785689  on  8 degrees of freedom
# Corrected Multiple R-squared:  0.9413342
```

Fitting the ANOVA table we find evidence of lack of fit:

```
model <- aov(Solids ~ -1 + (Monomer + Crosslinker + Resin)^2, data=df)
pe.summary(model)
#                         Df       Sum Sq     Mean Sq     F value
# Regression               6 19552.517756 3258.752959 102.7477558
#   Monomer                1  8892.687322 8892.687322 280.3844527
#   Crosslinker            1  4505.938320 4505.938320 142.0712327
#   Resin                  1  4787.555137 4787.555137 150.9505483
#   Monomer:Crosslinker    1     9.952813    9.952813   0.3138100
#   Monomer:Resin          1   320.492519  320.492519  10.1050578
#   Crosslinker:Resin      1  1035.891646 1035.891646  32.6614331
# Error                    8   267.386165   33.423271          NA
#   Lack of Fit            8   140.521965   17.565246   0.5538283
#   Pure Error             4   126.864200   31.716050          NA
#                             Pr(>F)
# Regression              2.472103e-04
```

```
#   Monomer               7.453962e-05
#   Crosslinker           2.838113e-04
#   Resin                 2.520814e-04
#   Monomer:Crosslinker   6.052398e-01
#   Monomer:Resin         3.357227e-02
#   Crosslinker:Resin     4.637189e-03
# Error                           NA
#   Lack of Fit           7.789473e-01
#   Pure Error                    NA
```

Since the quadratic model leaves no pure error we can't estimate lack of fit, so we'll stop here.

References

Dupuy, D., and C. Helbert. 2015. *DiceEval: Construction and Evaluation of Metamodels*. https://cran.r-project.org/package=DiceEval.

Helbert, Celine. 2018. *DiceDesign: Designs of Computer Experiments*. https://cran.r-project.org/package=DiceDesign.

Lawson, John. 2014. *Vdgraph: Variance Dispersion Graphs and Fraction of Design Space Plots for Response Surface Designs*. https://cran.r-project.org/package=Vdgraph.

Lawson, John, Cameron Willden, and Greg Piepel. 2016. *Mixexp: Design and Analysis of Mixture Experimentss*. https://cran.r-project.org/package=mixexp.

Lenth, Russel. 2018. *Rsm: Response-Surface Analysis*. https://cran.r-project.org/package=rsm.

Roth, Thomas. 2016. *QualityTools: Statistical Methods for Quality Science*. https://cran.r-project.org/package=qualityTools.

Chapter 12

Robust Parameter Design and Process Robustness Studies

Chapter 12 in the text introduces robust parameter designs. These designs allow the experimenter to investigate situations where not all of the factors can be easily controlled.

Example 12.1

Recall the filtration experiment from Example 6.2. Assume that temperature (`z`) is difficult to control, while pressure (`x1`), concentration (`x2`), and stirring rate (`x3`) are easy to control. We fit the following:

```
df <- Table6.10
colnames(df) <- c('z', 'x1', 'x2', 'x3', 'y', 'Block')
model <- lm(y ~ z * (x1 + x2 + x3), data=df)
print(summary(model))
#
# Call:
# lm(formula = y ~ z * (x1 + x2 + x3), data = df)
#
# Residuals:
#    Min     1Q Median     3Q    Max
# -6.000 -1.500  0.375  2.062  4.250
#
# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept)  70.0625     1.1040  63.464 4.23e-12 ***
# z            10.8125     1.1040   9.794 9.91e-06 ***
# x1            1.5625     1.1040   1.415 0.194695
# x2            4.9375     1.1040   4.472 0.002076 **
# x3            7.3125     1.1040   6.624 0.000165 ***
# z:x1          0.0625     1.1040   0.057 0.956241
```

```
# z:x2         -9.0625     1.1040  -8.209 3.63e-05 ***
# z:x3          8.3125     1.1040   7.530 6.74e-05 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 4.416 on 8 degrees of freedom
# Multiple R-squared:  0.9728,  Adjusted R-squared:  0.949
# F-statistic: 40.84 on 7 and 8 DF,  p-value: 1.219e-05
```

Since x_1 is not significant let's refit:

```
model <- lm(y ~ z * (x2 + x3), data=df)
print(summary(model))
#
# Call:
# lm(formula = y ~ z * (x2 + x3), data = df)
#
# Residuals:
#     Min      1Q  Median      3Q     Max
# -6.3750 -1.5000  0.0625  2.9062  5.7500
#
# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept)   70.062      1.104  63.444 2.30e-14 ***
# z             10.812      1.104   9.791 1.93e-06 ***
# x2             4.938      1.104   4.471   0.0012 **
# x3             7.313      1.104   6.622 5.92e-05 ***
# z:x2          -9.063      1.104  -8.206 9.41e-06 ***
# z:x3           8.312      1.104   7.527 2.00e-05 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 4.417 on 10 degrees of freedom
# Multiple R-squared:  0.966,   Adjusted R-squared:  0.9489
# F-statistic: 56.74 on 5 and 10 DF,  p-value: 5.14e-07
```

We can pick those terms that do not involve z to find $\mathbb{E}_z[y(\mathbf{x}, z)]$:

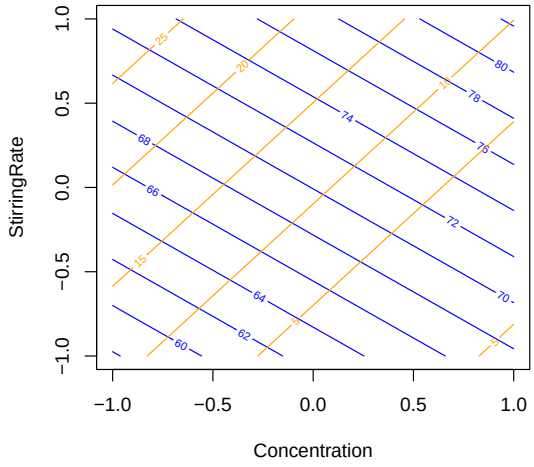
$$\mathbb{E}_z[y(\mathbf{x}, z)] = 70.06 + 4.94x_2 + 7.31x_3 \ .$$

Next, $\mathrm{Var}_z\big[y(\mathbf{x}, z)\big]$ is

$$\mathrm{Var}_z\big[y(\mathbf{x}, z)\big] = [10.812 - 9.063x_2 + 8.312x_3]^2\sigma_z^2 + \sigma^2 \ .$$

Holding temperature $z = 0$ we can plot the contours (blue) of the expected value, then plot the contours (orange) of the variance on top of that:

```
x2 <- seq(from=-1, to=1, length.out=100)
x3 <- seq(from=-1, to=1, length.out=100)
z <- matrix(predict(model, cbind(
    'z'=rep(0, 100^2),
    expand.grid('x2'=x2, 'x3'=x3)
)), ncol=100)

poe.coefs <- (
    model.matrix(
        ~ z * (x2 + x3),
        data=cbind(
            'z'=rep(1, 100^2),
            expand.grid('x2'=x2, 'x3'=x3)
        )
    )
    %*%
    matrix(c(0, coef(model)['z'], 0, 0,
↪   coef(model)[c('z:x2','z:x3')]))
)^2
z.poe <- matrix(sqrt(poe.coefs), nrow=100)

contour(x2, x3, z, col='blue', xlab='Concentration',
↪   ylab='StirringRate')
contour(x2, x3, z.poe, col='orange', add=TRUE)
```



Example 12.2

Consider the semiconductor manufacturing experiment data in Table 12.3. We fit the following model:

```
df <- Table12.3[,-1]
model <- lm(y ~ (z1 + z2 + z3) * (x1 + x2) + x1:x2 + I(x1^2) +
↪  I(x2^2), data=df)
print(summary(model))
#
# Call:
# lm(formula = y ~ (z1 + z2 + z3) * (x1 + x2) + x1:x2 + I(x1^2) +
#     I(x2^2), data = df)
#
# Residuals:
#      Min       1Q   Median       3Q      Max
# -1.15688 -0.43187  0.04292  0.38500  1.19292
#
# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept)  30.3650     0.4367  69.530 2.04e-12 ***
# z1            2.7312     0.2441  11.188 3.65e-06 ***
# z2           -2.3313     0.2441  -9.549 1.20e-05 ***
# z3            2.3313     0.2441   9.549 1.20e-05 ***
# x1           -2.9208     0.1993 -14.653 4.62e-07 ***
# x2           -4.1292     0.1993 -20.715 3.09e-08 ***
# I(x1^2)       2.5959     0.2217  11.707 2.59e-06 ***
# I(x2^2)       2.1834     0.2217   9.847 9.53e-06 ***
# z1:x1        -0.2687     0.2441  -1.101 0.302987
# z1:x2         2.0063     0.2441   8.218 3.60e-05 ***
# z2:x1         0.8938     0.2441   3.661 0.006393 **
# z2:x2        -1.4312     0.2441  -5.863 0.000377 ***
# z3:x1         2.5812     0.2441  10.573 5.59e-06 ***
# z3:x2         1.5562     0.2441   6.375 0.000215 ***
# x1:x2         2.8688     0.2441  11.751 2.52e-06 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 0.9765 on 8 degrees of freedom
# Multiple R-squared:  0.9948,  Adjusted R-squared:  0.9857
# F-statistic: 109.1 on 14 and 8 DF,  p-value: 1.508e-07
```

We can pick those terms that do not involve z_i to find $\mathbb{E}_z[y(\mathbf{x}, z)]$:

$$\mathbb{E}_z[y(\mathbf{x}, z)] = 30.365 - 2.921x_1 - 4.129x_2 + 2.869x_1x_2 + 2.596x_1^2 + 2.183x_2^2 \ .$$

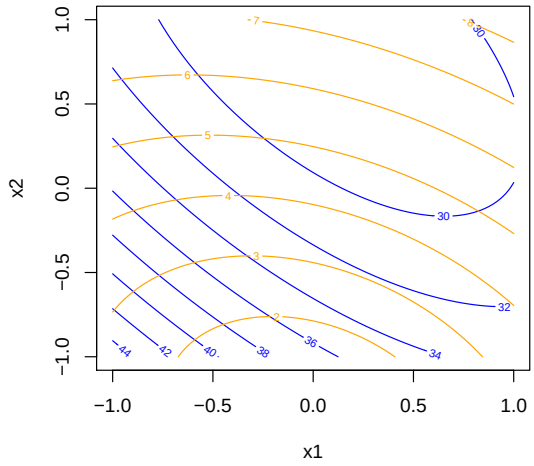
Next, $\text{Var}_z\big[y(\mathbf{x}, \mathbf{z})\big]$ is

$$\begin{aligned}
\text{Var}_{\mathbf{z}}\big[y(\mathbf{x}, \mathbf{z})\big] &= [2.731 - 0.269x_1 + 2.006x_2]^2\sigma_{z_1}^2 && (12.1)\\
&+ [-2.331 + 0.894x_1 - 1.431x_2]^2\sigma_{z_2}^2 && (12.2)\\
&+ [2.331 + 2.581x_1 + 1.556x_2]^2\sigma_{z_3}^2 && (12.3)\\
&+ \sigma^2 \ . && (12.4)
\end{aligned}$$

Letting $\sigma_{z_i}^2 = 1$ we plot the contours of mean (blue) and variance (orange) as follows:

```
x1 <- seq(from=-1, to=1, length.out=100)
x2 <- seq(from=-1, to=1, length.out=100)

mean.df <- cbind(
    data.frame(
        'z1'=rep(0, 100^2),
        'z2'=rep(0, 100^2),
        'z3'=rep(0, 100^2)
    ),
    expand.grid('x1'=x1, 'x2'=x2)
)

z <- matrix(predict(model, mean.df), ncol=100)

var.df <- cbind(
    data.frame(
        'z1'=rep(1, 100^2),
        'z2'=rep(1, 100^2),
        'z3'=rep(1, 100^2)
    ),
    expand.grid('x1'=x1, 'x2'=x2)
)

X.m <- model.matrix( ~ (z1 + z2 + z3) * (x1 + x2) + x1:x2 + I(x1^2) +
↪  I(x2^2), data=var.df)
b <- coef(model)
z1.terms <- matrix(c(0, b[2], 0, 0, 0, 0, 0, 0, b[9], b[10], 0, 0, 0,
↪  0, 0))
z2.terms <- matrix(c(0, 0, b[3], 0, 0, 0, 0, 0, 0, 0, b[11], b[12], 0,
↪  0, 0))
z3.terms <- matrix(c(0, 0, 0, b[4], 0, 0, 0, 0, 0, 0, 0, 0, b[13],
↪  b[14], 0))

poe.coefs <- sqrt((X.m %*% z1.terms)^2 + (X.m %*% z2.terms)^2 + (X.m
↪  %*% z3.terms)^2)
z.poe <- matrix(poe.coefs, nrow=100)
```

```
contour(x1, x2, z, col='blue', xlab='x1', ylab='x2')
contour(x1, x2, z.poe, col='orange', add=TRUE)
```



Chapter 13

Experiments with Random Factors

```
library(MontgomeryDAE)
options(contrasts=c('unordered'='contr.sum', 'ordered'='contr.poly'))
```

Chapter 13 introduces the distinction between mixed models where the interaction of a fixed and random effect must sum to zero (restricted) versus no such restriction (unrestricted). There are situations where there is little difference between the restricted and unrestricted models, and there are situations where the difference can be pronounced. Some confusion about this is likely added by the fact that the fitting technique "REML" is sometimes called *restricted maximum likelihood*, and thus talk of *restricted* versus *unrestricted* might refer to using REML versus maximum likelihood estimation. Here, as is the case in the text, the restricted model will refer to the requirement that the interaction of fixed and random effects must sum to zero.

There is no good way in R to fit the restricted model. Random and mixed effect models are often fit with either the `lme4` package or the `nlme` package. The `lme4` package (Bates et al. 2019) is more popular and easier to use, so I will focus on using that package here and in the remaining chapters, along with `lmerTest` (Kuznetsova, Brockhoff, and Christensen 2019). This means that our analysis will be unrestricted, and so our best hope is that the output will be close to that from Minitab and JMP though it need only be so when the text notes that the model is unrestricted.

Examples 13.1 Through 13.4 — A Measurement Systems Capability Study

Examples 13.1 through 13.4 refer to a measurement systems capability study where a gauge repeatability and reproducibility study was conducted on an instrument used to measure a critical part dimension in a manufacturing process. First, let's massage the table into a data frame appropriate for analysis:

```
df <- data.frame(
    'Part'=c(
        Table13.1$PartNumber, Table13.1$PartNumber,
↪  Table13.1$PartNumber
    ),
    'Operator'=factor(c(
        rep(1, 40), rep(2, 40), rep(3, 40)
    )),
    'Measurement'=c(
        Table13.1$Operator1, Table13.1$Operator2, Table13.1$Operator3
    )
)
```

Next, we'll use the `lme4` package to fit the model and the `lmerTest` package to test the significance of the variance components. The fixed effect's ANOVA table matches the text:

```
library(lme4)
library(lmerTest)

model <- lmer(Measurement ~ Operator + (1 | Part) + (1 |
↪  Part:Operator), data=df, REML=TRUE)
print(anova(model))
#> Type III Analysis of Variance Table with Satterthwaite's method
#>          Sum Sq Mean Sq NumDF DenDF F value Pr(>F)
#> Operator 2.6167  1.3083     2    98  1.4814 0.2324
```

The variance component for `Part` is near to that from the restricted model:

```
print(summary(model))
#> Linear mixed model fit by REML. t-tests use Satterthwaite's method
↪  [
#> lmerModLmerTest]
#> Formula: Measurement ~ Operator + (1 | Part) + (1 | Part:Operator)
#>    Data: df
```

```
#>
#> REML criterion at convergence: 411.7
#>
#> Scaled residuals:
#>     Min      1Q  Median      3Q     Max
#> -2.1809 -0.7183  0.1445  0.5253  2.5849
#>
#> Random effects:
#>  Groups        Name        Variance Std.Dev.
#>  Part:Operator (Intercept)  0.0000  0.0000
#>  Part          (Intercept) 10.2513  3.2018
#>  Residual                   0.8832  0.9398
#> Number of obs: 120, groups:  Part:Operator, 60; Part, 20
#>
#> Fixed effects:
#>             Estimate Std. Error       df t value Pr(>|t|)
#> (Intercept) 22.39167    0.72106 19.00002  31.054   <2e-16 ***
#> Operator1   -0.09167    0.12132 97.99999  -0.756    0.452
#> Operator2   -0.11667    0.12132 97.99999  -0.962    0.339
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Correlation of Fixed Effects:
#>           (Intr) Oprtr1
#> Operator1  0.000
#> Operator2  0.000 -0.500
#> optimizer (nloptwrap) convergence code: 0 (OK)
#> boundary (singular) fit: see ?isSingular
```

The significance of the `Part` random effect is extreme, as it is in the restricted model:

```
print(rand(model))
#> ANOVA-like table for random-effects: Single term deletions
#>
#> Model:
#> Measurement ~ Operator + (1 | Part) + (1 | Part:Operator)
#>                    npar  logLik    AIC    LRT Df Pr(>Chisq)
#> <none>                6 -205.83 423.65
#> (1 | Part)            5 -259.53 529.06 107.41  1     <2e-16 ***
#> (1 | Part:Operator)   5 -205.83 421.65   0.00  1          1
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Example 13.5

In Example 13.5 an engineer is investigating the effect of inlet side, operator, and specific gauge used by the operator on the pressure drop measures across an expansion valve within a turbine. First we'll make our data frame with `GasTemperature` as a factor:

```
df <- Table13.9
df[,'GasTemperature'] <- factor(df[,'GasTemperature'])
colnames(df) <- c('A','B','C','PressureDrop')
```

Next, we'll fit the model:

```
library(lme4)
library(lmerTest)

model <- lmer(PressureDrop ~ A + (1|B) + (1|C) + (1|A:B) + (1|A:C) +
↪  (1|B:C) + (1|A:B:C), data=df, REML=TRUE)
print(summary(model))
#> Linear mixed model fit by REML. t-tests use Satterthwaite's method
↪  [
#> lmerModLmerTest]
#> Formula: PressureDrop ~ A + (1 | B) + (1 | C) + (1 | A:B) + (1 |
↪  A:C) +
#>     (1 | B:C) + (1 | A:B:C)
#>    Data: df
#>
#> REML criterion at convergence: 438.1
#>
#> Scaled residuals:
#>     Min      1Q  Median      3Q     Max
#> -1.7201 -0.6468 -0.1159  0.6592  4.0073
#>
#> Random effects:
#>  Groups   Name        Variance  Std.Dev.
#>  A:B:C    (Intercept) 4.917e-09 7.012e-05
#>  B:C      (Intercept) 1.268e+00 1.126e+00
#>  A:B      (Intercept) 2.674e+01 5.171e+00
#>  A:C      (Intercept) 7.072e-01 8.410e-01
#>  B        (Intercept) 2.667e-08 1.633e-04
#>  C        (Intercept) 0.000e+00 0.000e+00
#>  Residual             1.999e+01 4.471e+00
#> Number of obs: 72, groups:
#> A:B:C, 36; B:C, 12; A:B, 12; A:C, 9; B, 4; C, 3
#>
#> Fixed effects:
#>             Estimate Std. Error      df t value Pr(>|t|)
#> (Intercept)   0.3472     1.6403  9.5184   0.212   0.8368
```

```
#> A1           -2.9722     2.2737  9.0821  -1.307   0.2233
#> A2            5.3194     2.2737  9.0821   2.340   0.0438 *
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Correlation of Fixed Effects:
#>    (Intr) A1
#> A1  0.000
#> A2  0.000 -0.500
#> optimizer (nloptwrap) convergence code: 0 (OK)
#> boundary (singular) fit: see ?isSingular
print(rand(model))
#> ANOVA-like table for random-effects: Single term deletions
#>
#> Model:
#> PressureDrop ~ A + (1 | B) + (1 | C) + (1 | A:B) + (1 | A:C) + (1 |
↪  B:C) + (1 | A:B:C)
#>             npar  logLik    AIC     LRT Df Pr(>Chisq)
#> <none>        10 -219.05 458.10
#> (1 | B)        9 -219.05 456.10  0.0000  1     1.0000
#> (1 | C)        9 -219.05 456.10  0.0000  1     0.9997
#> (1 | A:B)      9 -227.19 472.37 16.2713  1  5.489e-05 ***
#> (1 | A:C)      9 -219.13 456.26  0.1604  1     0.6888
#> (1 | B:C)      9 -219.23 456.46  0.3602  1     0.5484
#> (1 | A:B:C)    9 -219.05 456.10  0.0000  1     1.0000
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We arrive at the same conclusion as the text, that the `GasTemperature:Operator` interaction is the only significant variance component.

References

Bates, Douglas, Marint Maechler, Ben Bolker, Steven Walker, Christensen Rune Jaubo Bojesen, Henrik Singmann, Bin Dai, et al. 2019. *Lme4: Linear Mixed-Effects Models Using 'Eigen' and S4*. https://cran.r-project.org/package=lme4.

Kuznetsova, Alexandra, Per Bruun Brockhoff, and Rune Jaubo Bojesen Christensen. 2019. *LmerTest: Tests in Linear Mixed Effects Models*. https://cran.r-project.org/package=lmerTest.

Chapter 14

Nested and Split-Plot Designs

Note: The unrestricted model is used for mixed effects. For more information see the note at beginning of this guide for Chapter 13.

Nested and split-plot designs are common in industrial settings. One of the most common mistakes I've encountered in the field is experimenters assuming their experiment is not a split-plot when it really is. Chapter 13 of the text introduces the nested and split-plot models.

Example 14.1

In this example a company that buys material in batches from three different suppliers wishes to better understand the variable purity of the material from each supplier. Since each batch of material came from one supplier this will be a nested random effects experiment. The way we model a nested effect in `lme4` is with the expression `(1 | Parent/Child)` for a child effect nested under a parent:

```
library(lme4)
library(lmerTest)

model <- lmer(Purity ~ (1 | Supplier/Batches), data=Table14.3)
print(summary(model))
# Linear mixed model fit by REML. t-tests use Satterthwaite's method [
# lmerModLmerTest]
# Formula: Purity ~ (1 | Supplier/Batches)
#    Data: Table14.3
#
# REML criterion at convergence: 148.7
#
# Scaled residuals:
#      Min       1Q   Median       3Q      Max
```

```
# -1.38226 -0.75533 -0.07592  0.57348  1.71092
#
# Random effects:
#  Groups           Name        Variance  Std.Dev.
#  Batches:Supplier (Intercept) 1.696e+00 1.302285
#  Supplier         (Intercept) 9.197e-07 0.000959
#  Residual                     2.639e+00 1.624383
# Number of obs: 36, groups:  Batches:Supplier, 12; Supplier, 3
#
# Fixed effects:
#             Estimate Std. Error      df t value Pr(>|t|)
# (Intercept)   0.3611     0.4633 10.9924   0.779    0.452
print(rand(model))
# ANOVA-like table for random-effects: Single term deletions
#
# Model:
# Purity ~ (1 | Batches:Supplier) + (1 | Supplier)
#                        npar  logLik    AIC    LRT Df Pr(>Chisq)
# <none>                    4 -74.343 156.69
# (1 | Batches:Supplier)    3 -76.503 159.00 4.3186  1     0.0377 *
# (1 | Supplier)            3 -74.343 154.69 0.0000  1     1.0000
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here we find the majority of the variation is from batch-to-batch within each supplier, so it's unlikely that we'll be able to solve this problem by picking the "best" supplier.

Example 14.2

An engineer is studying manual insertion of components on printed circuit boards using three different fixtures and two different layouts. Four operators are selected at random to participate in the experiment:

```
df <- Table14.10
df['FO'] <- factor(sprintf('F%s.O%s', as.character(df['Fixture']),
↪  as.character(df['Operator'])))
model <- lmer(AssemblyTime ~ Layout * Fixture + (1|Operator/Layout) +
↪  (1|FO/Layout), data=df)
# boundary (singular) fit: see ?isSingular
print(anova(model))
# Type III Analysis of Variance Table with Satterthwaite's method
#               Sum Sq Mean Sq NumDF   DenDF F value   Pr(>F)
# Layout         0.795  0.7949     1  5.9998  0.3407 0.580709
# Fixture       35.214 17.6071     2 12.0005  7.5458 0.007552 **
```

```
# Layout:Fixture  8.099  4.0496     2 12.0005  1.7355 0.217758
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

When we look at the REML fit we find that one of the random effect estimates is identically zero, and another is very near zero. The remaining effects are within about one standard deviation of zero. It appears that R is having trouble dealing with the boundary in this model. In practice it would be a good idea to remove those random effects and refit the model, however, the model we have is as follows:

```
print(summary(model))
# Linear mixed model fit by REML. t-tests use Satterthwaite's method [
# lmerModLmerTest]
# Formula: AssemblyTime ~ Layout * Fixture + (1 | Operator/Layout) +
↪  (1 |
#     FO/Layout)
#    Data: df
#
# REML criterion at convergence: 195.9
#
# Scaled residuals:
#      Min       1Q   Median       3Q      Max
# -1.47784 -0.69539 -0.04366  0.70206  1.37328
#
# Random effects:
#  Groups          Name        Variance  Std.Dev.
#  Layout:FO       (Intercept) 1.576e+00 1.256e+00
#  FO              (Intercept) 1.622e-07 4.028e-04
#  Layout:Operator (Intercept) 1.083e+00 1.041e+00
#  Operator        (Intercept) 7.031e-10 2.652e-05
#  Residual                    2.333e+00 1.528e+00
# Number of obs: 48, groups:
# Layout:FO, 24; FO, 12; Layout:Operator, 8; Operator, 4
#
# Fixed effects:
#                 Estimate Std. Error      df t value Pr(>|t|)
# (Intercept)      26.0833     0.4997  5.9998  52.196 3.32e-09 ***
# Layout1          -0.2917     0.4997  5.9998  -0.584  0.58071
# Fixture1         -0.8333     0.4781 12.0005  -1.743  0.10687
# Fixture2          1.8542     0.4781 12.0005   3.878  0.00219 **
# Layout1:Fixture1 -0.2083     0.4781 12.0005  -0.436  0.67075
# Layout1:Fixture2  0.8542     0.4781 12.0005   1.787  0.09927 .
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Correlation of Fixed Effects:
#             (Intr) Layot1 Fixtr1 Fixtr2 Ly1:F1
```

```
# Layout1      0.000
# Fixture1     0.000  0.000
# Fixture2     0.000  0.000 -0.500
# Layt1:Fxtr1  0.000  0.000  0.000  0.000
# Layt1:Fxtr2  0.000  0.000  0.000  0.000 -0.500
# optimizer (nloptwrap) convergence code: 0 (OK)
# boundary (singular) fit: see ?isSingular
```

When we go to calculate the significance of the random effects we are again warned of the problem with fitting, and find none to be significant:

```
print(rand(model))
# boundary (singular) fit: see ?isSingular
# boundary (singular) fit: see ?isSingular
# boundary (singular) fit: see ?isSingular
# boundary (singular) fit: see ?isSingular
# ANOVA-like table for random-effects: Single term deletions
#
# Model:
# AssemblyTime ~ Layout + Fixture + (1 | Layout:Operator) + (1 |
↪  Operator) + (1 | Layout:FO) + (1 | FO) + Layout:Fixture
#                       npar  logLik    AIC    LRT Df Pr(>Chisq)
# <none>                   11 -97.943 217.88
# (1 | Layout:Operator)   10 -98.594 217.19 1.3021  1     0.2538
# (1 | Operator)          10 -97.943 215.88 0.0000  1     1.0000
# (1 | Layout:FO)         10 -98.825 217.65 1.7648  1     0.1840
# (1 | FO)                10 -97.943 215.88 0.0000  1     1.0000
```

Section 14.4 — The Split-Plot Design

This section discusses the experiment on the tensile strength of paper in Table 14.16. First let's fit it using fixed effects:

```
df <- Table14.16
df['Temperature'] <- factor(df['Temperature'])
model <- aov(TensileStrength ~ Temperature * Preperation +
↪  Error(Replicate), data=df)
summary(model)
#
# Error: Replicate
#           Df Sum Sq Mean Sq F value Pr(>F)
# Residuals  2  77.56   38.78
#
# Error: Within
#                         Df Sum Sq Mean Sq F value   Pr(>F)
```

```
# Temperature              3  434.1  144.69  29.536 6.74e-08 ***
# Preperation              2  128.4   64.19  13.104 0.000179 ***
# Temperature:Preperation  6   75.2   12.53   2.557 0.049434 *
# Residuals               22  107.8    4.90
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Next, let's use the mixed effects model:

```
library(lme4)
library(lmerTest)

df <- Table14.16
df['Temperature'] <- factor(df['Temperature'])
model <- lmer(TensileStrength ~ Temperature * Preperation *
↪  (1|Replicate), data=df)
print(anova(model))
# Type III Analysis of Variance Table with Satterthwaite's method
#                         Sum Sq Mean Sq NumDF DenDF F value    Pr(>F)
↪
# Temperature             434.08 144.694     3    22 29.5356 6.743e-08
↪  ***
# Preperation             128.39  64.194     2    22 13.1036 0.0001788
↪  ***
# Temperature:Preperation  75.17  12.528     6    22  2.5572 0.0494335
↪  *
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
print(summary(model))
# Linear mixed model fit by REML. t-tests use Satterthwaite's method [
# lmerModLmerTest]
# Formula: TensileStrength ~ Temperature * Preperation * (1 |
↪  Replicate)
#    Data: df
#
# REML criterion at convergence: 140.7
#
# Scaled residuals:
#     Min      1Q  Median      3Q     Max
# -1.4534 -0.5615 -0.1337  0.6199  1.4080
#
# Random effects:
#  Groups    Name        Variance Std.Dev.
#  Replicate (Intercept) 2.823    1.680
#  Residual              4.899    2.213
# Number of obs: 36, groups:  Replicate, 3
#
# Fixed effects:
#                          Estimate Std. Error      df t value
↪  Pr(>|t|)
# (Intercept)               36.0278     1.0379  2.0000  34.713
↪  0.000829 ***
```

```
# Temperature1                -4.8056     0.6389 22.0000  -7.521
↪  1.62e-07 ***
# Temperature2                -1.4722     0.6389 22.0000  -2.304
↪  0.031035 *
# Temperature3                 1.8611     0.6389 22.0000   2.913
↪  0.008065 **
# Preperation1                -0.3611     0.5217 22.0000  -0.692
↪  0.496061
# Preperation2                 2.4722     0.5217 22.0000   4.739
↪  9.93e-05 ***
# Temperature1:Preperation1  -1.1944     0.9036 22.0000  -1.322
↪  0.199793
# Temperature2:Preperation1   0.4722     0.9036 22.0000   0.523
↪  0.606476
# Temperature3:Preperation1   1.8056     0.9036 22.0000   1.998
↪  0.058207 .
# Temperature1:Preperation2  -0.3611     0.9036 22.0000  -0.400
↪  0.693279
# Temperature2:Preperation2   1.9722     0.9036 22.0000   2.183
↪  0.040023 *
# Temperature3:Preperation2  -0.6944     0.9036 22.0000  -0.769
↪  0.450345
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Correlation of Fixed Effects:
#             (Intr) Tmprt1 Tmprt2 Tmprt3 Prprt1 Prprt2 Tm1:P1 Tm2:P1
↪  Tm3:P1
# Temperatur1  0.000
↪
# Temperatur2  0.000 -0.333
↪
# Temperatur3  0.000 -0.333 -0.333
↪
# Preperatin1  0.000  0.000  0.000  0.000
↪
# Preperatin2  0.000  0.000  0.000  0.000 -0.500
↪
# Tmprtr1:Pr1  0.000  0.000  0.000  0.000  0.000  0.000
↪
# Tmprtr2:Pr1  0.000  0.000  0.000  0.000  0.000  0.000 -0.333
↪
# Tmprtr3:Pr1  0.000  0.000  0.000  0.000  0.000  0.000 -0.333 -0.333
↪
# Tmprtr1:Pr2  0.000  0.000  0.000  0.000  0.000  0.000 -0.500  0.167
↪  0.167
# Tmprtr2:Pr2  0.000  0.000  0.000  0.000  0.000  0.000  0.167 -0.500
↪  0.167
# Tmprtr3:Pr2  0.000  0.000  0.000  0.000  0.000  0.000  0.167  0.167
↪  -0.500
#             Tm1:P2 Tm2:P2
# Temperatur1
```

```
# Temperatur2
# Temperatur3
# Preperatin1
# Preperatin2
# Tmprtr1:Pr1
# Tmprtr2:Pr1
# Tmprtr3:Pr1
# Tmprtr1:Pr2
# Tmprtr2:Pr2 -0.333
# Tmprtr3:Pr2 -0.333 -0.333
print(rand(model))
# ANOVA-like table for random-effects: Single term deletions
#
# Model:
# TensileStrength ~ Temperature + Preperation + (1 | Replicate) +
↪  Temperature:Preperation
#                 npar  logLik    AIC    LRT Df Pr(>Chisq)
# <none>            14 -70.337 168.67
# (1 | Replicate)   13 -73.729 173.46 6.7841  1   0.009197 **
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note that above we find R has converged on a solution, so we feel good about the validity of these results.

Example 14.3

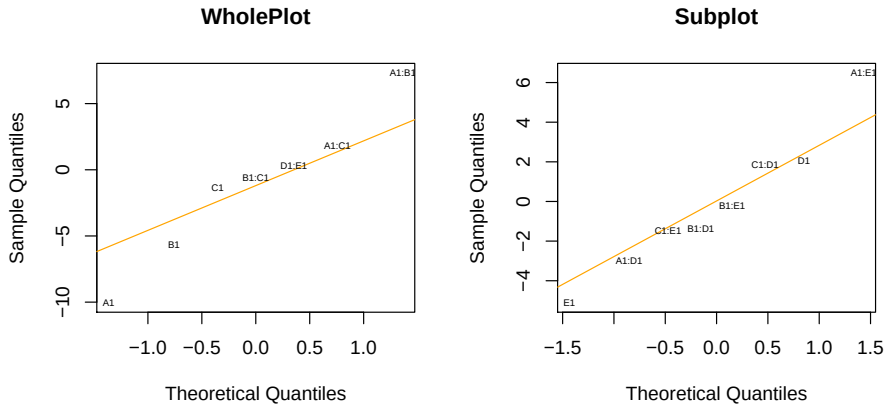
A single wafer plasma etching process is being investigated to understand the effect of electrode gap (`A`), gas flow (`B`), pressure (`C`), time (`D`), and RF power (`E`). Note that `A`, `B`, and `C` are difficult to change run-to-run, but `D` and `E` are easy to change. The data is in Table 14.22. We fit the coefficient estimates using the regular `aov` function as `lmer` does not well handle this example:

```
model <- aov(Uniformity ~ (A + B + C + D + E)^2 +  Error(WholePlot),
↪  data=Table14.22)
coef(model)
# (Intercept) :
# (Intercept)
#    49.74188
#
# WholePlot :
#         A1         B1         C1      A1:B1      A1:C1      B1:C1
# -10.065625  -5.624375  -1.328125   7.343125   1.834375  -0.604375
#      D1:E1
```

```
#   0.336875
#
# Within :
#        D1        E1     A1:D1     A1:E1     B1:D1     B1:E1
↪  C1:D1
#  2.069375 -5.123125 -3.005625  6.501875 -1.361875 -0.209375
↪  1.869375
#     C1:E1
# -1.488125
```

We have to construct our own normal probability plots because `halfnormal` from `DoE.base` doesn't understand the error structure of the model:

```
op <- par(mfrow=c(1,2))
# Whole plot
labs <- names(coef(model)$WholePlot)
pts <-qqnorm(coef(model)$WholePlot, col='white', main='WholePlot')
qqline(pts$y, col='orange')
text(x=pts$x, y=pts$y, labels=labs, cex=0.5)
# Sub plot
labs <- names(coef(model)$Within)
pts <-qqnorm(coef(model)$Within, col='white', main='Subplot')
qqline(pts$y, col='orange')
text(x=pts$x, y=pts$y, labels=labs, cex=0.5)
```




```
par(op)
```

The probability plots indicate `A`, `B` and `A:B` are good candidates for the whole plot active effects, and `E` and `A:E` are good subplot active effect candidates.

Chapter 15

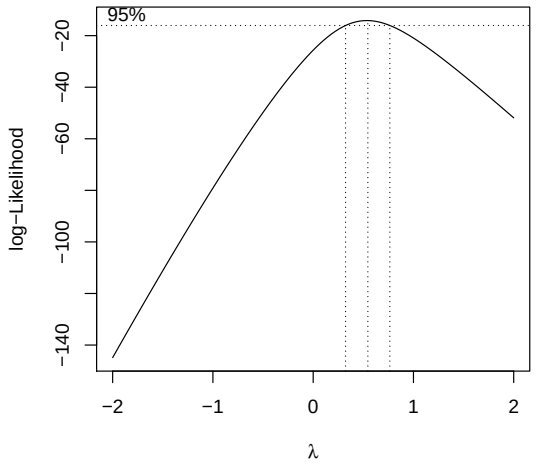
Other Design and Analysis Topics

Chapter 15 in the text and the associated supplemental material covers several important methods and design concerns. This includes repeated measures, normality-inducing transformations, unbalanced data problems, and analysis of covariance.

Example 15.1 — Box-Cox Transformation

Using the data from `Example3.5` we can find the Box-Cox transform of the response to make our data look more normally distributed:

```
library(MASS)

model <- lm(Observation ~ EstimationMethod, data=Example3.5)
vals <- boxcox(model)
```



The plot indicates that the 95% confidence interval doesn't contain 0 (so no log transform), and that the best is somewhere between around 1/2 to 3/4. By assigning the value returned by `boxcox` to a variable we can find the peak:

```
best.lambda <- vals$x[which.max(vals$y)]
print(best.lambda)
# [1] 0.5454545
```

This suggests 0.5 is an appropriate value for λ.

Example 15.2

Recall the coupon redemption data from Table 15.1, where the number of coupons redeemed is out of 1000 customers. To fit a binomial GLM (logistic regression) we use the `glm` function and indicate the family is `binomial`. Note that the response won't be a number, but will be two columns: the first column is the number of successes and the second is the number of failures:

```
model <- glm(cbind(Coupons, I(1000 - Coupons)) ~ (A + B + C)^2,
↪  data=Table15.1, family='binomial')
print(summary(model))
#
# Call:
# glm(formula = cbind(Coupons, I(1000 - Coupons)) ~ (A + B + C)^2,
#     family = "binomial", data = Table15.1)
#
```

```
# Deviance Residuals:
#       1        2        3        4        5        6        7
↪  8
#  0.4723  -0.4307  -0.4228   0.3949  -0.4572   0.4166   0.4238
↪  -0.3987
#
# Coefficients:
#              Estimate Std. Error z value Pr(>|z|)
# (Intercept) -1.011545   0.025515 -39.645  < 2e-16 ***
# A            0.169208   0.025509   6.633 3.28e-11 ***
# B            0.169622   0.025515   6.648 2.97e-11 ***
# C            0.023317   0.025510   0.914    0.361
# A:B         -0.006285   0.025512  -0.246    0.805
# A:C         -0.002773   0.025432  -0.109    0.913
# B:C         -0.041020   0.025434  -1.613    0.107
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# (Dispersion parameter for binomial family taken to be 1)
#
#     Null deviance: 93.0238  on 7  degrees of freedom
# Residual deviance:  1.4645  on 1  degrees of freedom
# AIC: 72.286
#
# Number of Fisher Scoring iterations: 3
```

Example 15.3 — The Grill Defects Experiment

The grill defects experiment described in problem 8.51 is analyzed in the text using Poisson regression. First we retrieve the number of defects.

```
df <- Problem8.51[,c('A','B','C','D','E','F','G','H','J')]
df[,'Defects'] <- round(Problem8.51[,'Sqrt']^2)
```

Next, we fit a poisson model:

```
model <- glm(Defects ~ D + F + B:G, data=df, family='poisson')
summary(model)
#
# Call:
# glm(formula = Defects ~ D + F + B:G, family = "poisson", data = df)
#
```

```
# Deviance Residuals:
#     Min       1Q   Median       3Q      Max
# -3.5212  -0.6105   0.0113   0.7059   1.4691
#
# Coefficients:
#             Estimate Std. Error z value Pr(>|z|)
# (Intercept)   1.1284     0.1734   6.508 7.60e-11 ***
# D            -0.8959     0.1126  -7.956 1.78e-15 ***
# F            -1.1757     0.1398  -8.407  < 2e-16 ***
# B:G          -0.7370     0.1012  -7.283 3.28e-13 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# (Dispersion parameter for poisson family taken to be 1)
#
#     Null deviance: 313.161  on 15  degrees of freedom
# Residual deviance:  25.609  on 12  degrees of freedom
# AIC: 80.246
#
# Number of Fisher Scoring iterations: 5
```

Next, we generate approximate 95% confidence intervals around each observation:

```
response <- predict(model, type='response', se.fit=TRUE)
2 * qnorm(0.975) * response$se.fit
#          1          2          3          4          5          6
# 19.3364693  8.6039941  1.3830047  5.2018322  1.3830047  5.2018322
#          7          8          9         10         11         12
# 19.3364693  8.6039941  1.0584072  0.2691856  2.0463419  7.1430108
#         13         14         15         16
#  2.0463419  7.1430108  1.0584072  0.2691856
```

Example 15.4 — The Worsted Yarn Experiment

Here we fit a Gamma GLM to the data. Note that `Gamma` is capitalized in the family argument, and that we pass `'log'` as the link function:

```
model <- glm(CyclesToFailure ~ x1 + x2 + x3, data=Table15.4,
↪  family=Gamma(link='log'))
summary(model)
#
```

```
# Call:
# glm(formula = CyclesToFailure ~ x1 + x2 + x3, family = Gamma(link =
↪  "log"),
#     data = Table15.4)
#
# Deviance Residuals:
#      Min        1Q    Median        3Q       Max
# -0.43391  -0.11553  -0.00922   0.10260   0.25342
#
# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept)  6.34891    0.03422 185.511  < 2e-16 ***
# x1           0.84251    0.04192  20.100 4.34e-16 ***
# x2          -0.63132    0.04192 -15.062 2.10e-13 ***
# x3          -0.38513    0.04192  -9.188 3.68e-09 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# (Dispersion parameter for Gamma family taken to be 0.0316243)
#
#     Null deviance: 22.88613  on 26  degrees of freedom
# Residual deviance:  0.76939  on 23  degrees of freedom
# AIC: 332.76
#
# Number of Fisher Scoring iterations: 5
```

Next, we generate approximate 95% confidence intervals around each observation:

```
response <- predict(model, type='response', se.fit=TRUE)
2 * qnorm(0.975) * response$se.fit
#          1          2          3          4          5          6
#  214.10607  124.22701   99.10732   97.11774   52.23685   44.95472
#          7          8          9         10         11         12
#   60.57185   35.14454   28.03804  424.00889  228.06223  196.26901
#         13         14         15         16         17         18
#  178.29366   76.71924   82.53016  119.95458   64.52013   55.52564
#         19         20         21         22         23         24
# 1154.57899  669.90110  534.44181  523.71285  281.69013  242.42087
#         25         26         27
#  326.63710  189.51891  151.19669
```

Example 15.5

Note that by placing `I(x - mean(x))` before `Machine` we cause machine to be adjusted by the covariate:

```
model <- aov(y ~ I(x - mean(x)) + Machine, data=Table15.10)
print(summary(model))
#                Df Sum Sq Mean Sq F value   Pr(>F)
# I(x - mean(x))  1 305.13  305.13 119.933 2.96e-07 ***
# Machine         2  13.28    6.64   2.611    0.118
# Residuals      11  27.99    2.54
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Table 15.17

We reproduce the analysis from JMP given in Table 15.17 for the data set in Table 15.16. First, we'll fit the ugly full model as a regression model:

```
op <- options(contrast=c('contr.sum', 'contr.poly'))
model <- lm(y ~ I(x - mean(x)) * (A + B + C)^2, data=Table15.16)
options(op)
print(summary(model))
#
# Call:
# lm(formula = y ~ I(x - mean(x)) * (A + B + C)^2, data = Table15.16)
#
# Residuals:
#         1         2         3         4         5         6
↪  7
# -0.652000  0.473034 -0.317125  0.145270  0.009703  0.569189
↪  0.161177
#         8         9        10        11        12        13
↪  14
#  0.110712  0.905002 -0.726035  0.064123  0.107732 -0.262704
↪  -0.316187
#        15        16
#  0.091824 -0.363713
#
# Coefficients:
#                    Estimate Std. Error t value Pr(>|t|)
# (Intercept)         21.39218    1.29005  16.582  0.00362 **
# I(x - mean(x))       2.14202    0.36187   5.919  0.02737 *
# A                   13.88759    1.29103  10.757  0.00853 **
# B                   19.44430    1.12642  17.262  0.00334 **
# C                    5.09080    1.11486   4.566  0.04476 *
# A:B                -19.59556    0.70141 -27.937  0.00128 **
# A:C                 -0.23434    1.25916  -0.186  0.86953
# B:C                 -0.36807    0.88768  -0.415  0.71865
# I(x - mean(x)):A     2.11714    0.42977   4.926  0.03882 *
# I(x - mean(x)):B     3.01754    0.36471   8.274  0.01430 *
# I(x - mean(x)):C    -0.09672    0.35689  -0.271  0.81180
```

```
# I(x - mean(x)):A:B  -2.94911    0.19028 -15.498  0.00414 **
# I(x - mean(x)):A:C  -0.01212    0.41587  -0.029  0.97940
# I(x - mean(x)):B:C   0.08482    0.40200   0.211  0.85244
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 1.184 on 2 degrees of freedom
# Multiple R-squared:  0.9999,  Adjusted R-squared:  0.999
# F-statistic:  1206 on 13 and 2 DF,  p-value: 0.0008286
```

Next, we'll fit the reduced model as an ANOVA model:

```
xx <- (Table15.16$x - mean(Table15.16$x))
op <- options(contrast=c('contr.sum', 'contr.poly'))
model <- aov(y ~ xx + A + B + C + A:B + A:xx + B:xx + A:B:xx,
↪  data=Table15.16)
options(op)
print(summary(model))
#             Df Sum Sq Mean Sq  F value   Pr(>F)
# xx           1  12156   12156 15068.03 6.28e-13 ***
# A            1   1321    1321  1637.04 1.47e-09 ***
# B            1   3998    3998  4955.30 3.07e-11 ***
# C            1     53      53    65.38 8.52e-05 ***
# A:B          1   3788    3788  4695.81 3.70e-11 ***
# xx:A         1     64      64    78.92 4.64e-05 ***
# xx:B         1     65      65    80.13 4.42e-05 ***
# xx:A:B       1    543     543   672.90 3.24e-08 ***
# Residuals    7      6       1
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```