In-class Examples with R Code
Response Surface Analysis (RSM)
Stat 579
University of New Mexico

Erik B. Erhardt

Fall 2014

Contents

Chapter 1

Introduction

1.1 Introduction to R

Please see my course notes for ADA1 Ch 0 and ADA2 Ch 1 or other online resources for getting started with R.

Chapter 2

Building Empirical Models

2.1 Table 2.8, p. 48

2.1.1 Read data

Read data, skip extra header lines, recode variables.

```
#### 2.8
fn.data <- "http://statacumen.com/teach/RSM/data/RSM_TAB_02-08.txt"
df.2.8 <- read.table(fn.data, header=TRUE, skip=1)
str(df.2.8)
```

```
## 'data.frame': 12 obs. of  3 variables:
##  $ x1: int  -1 1 -1 1 -9 9 0 0 0 0 ...
##  $ x2: int  -1 -1 1 1 0 0 -9 9 0 0 ...
##  $ y : int  43 78 69 73 48 76 65 74 76 79 ...
```

```
df.2.8
```

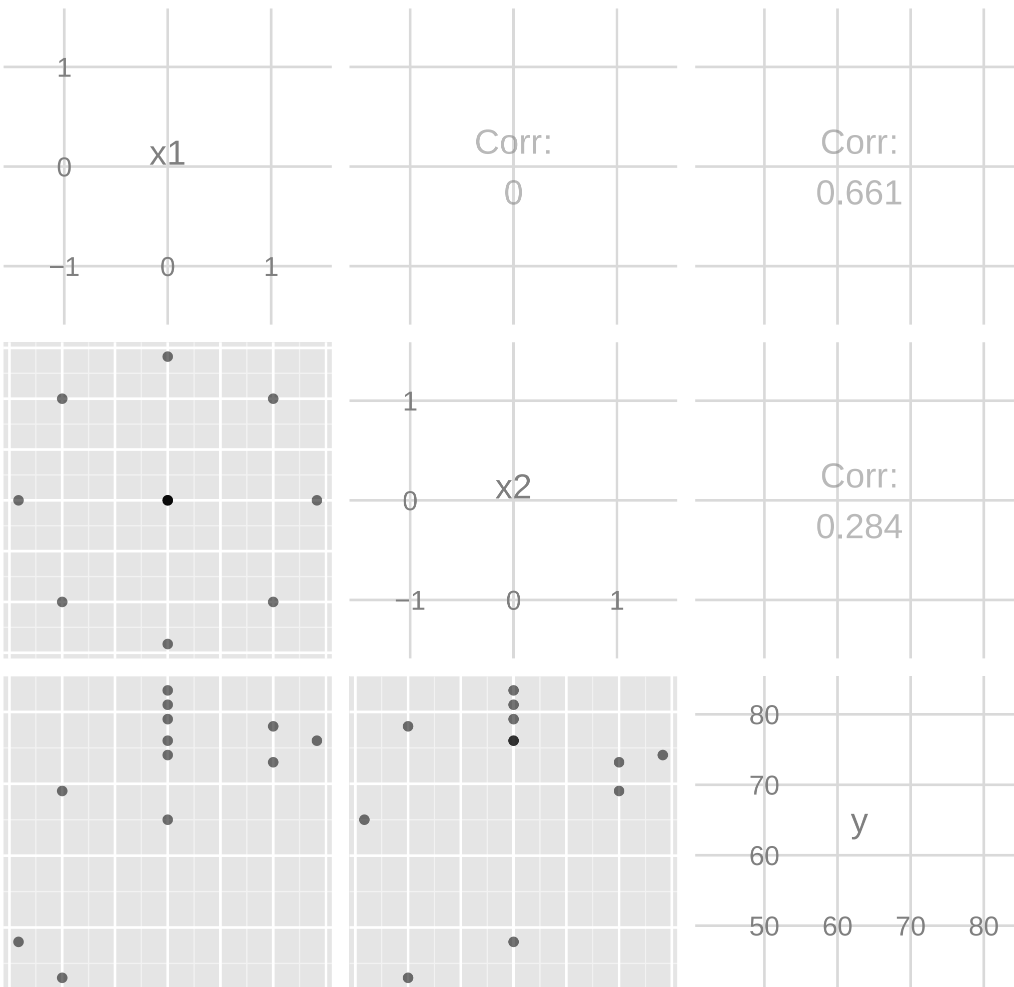
```
##    x1 x2  y
## 1  -1 -1 43
## 2   1 -1 78
## 3  -1  1 69
## 4   1  1 73
## 5  -9  0 48
## 6   9  0 76
## 7   0 -9 65
## 8   0  9 74
## 9   0  0 76
## 10  0  0 79
## 11  0  0 83
## 12  0  0 81
```

```
# replace coded values "9" with sqrt(2)
 # if x1= 9 then x1= sqrt(2)
df.2.8[,c("x1","x2")] <- replace(df.2.8[,c("x1","x2")], (df.2.8[,c("x1","x2")] ==  9)
                                , sqrt(2))
df.2.8[,c("x1","x2")] <- replace(df.2.8[,c("x1","x2")], (df.2.8[,c("x1","x2")] == -9)
                                , -sqrt(2))
df.2.8
```

```
##        x1     x2  y
## 1  -1.000 -1.000 43
## 2   1.000 -1.000 78
## 3  -1.000  1.000 69
## 4   1.000  1.000 73
## 5  -1.414  0.000 48
## 6   1.414  0.000 76
## 7   0.000 -1.414 65
## 8   0.000  1.414 74
## 9   0.000  0.000 76
## 10  0.000  0.000 79
## 11  0.000  0.000 83
## 12  0.000  0.000 81
```

Scatterplot matrix shows some relationships between y and other variables.

```
library(ggplot2)
suppressMessages(suppressWarnings(library(GGally)))
p <- ggpairs(df.2.8, alpha = 0.1)
# put scatterplots on top so y axis is vertical
#p <- ggpairs(df.2.8, upper = list(continuous = "points")
#                 , lower = list(continuous = "cor")
#             )
print(p)

  # detach package after use so reshape2 works (old reshape (v.1) conflicts)
  detach("package:GGally", unload=TRUE)
  detach("package:reshape", unload=TRUE)

## Error:  invalid 'name' argument
```



Correlation matrix indicates some (linear) correlations with y are different than zero, but if curvature exists, this summary is not very meaningful.

```
# correlation matrix and associated p-values testing "H0: rho == 0"
library(Hmisc)
rcorr(as.matrix(df.2.8))

##      x1   x2    y
## x1 1.00 0.00 0.66
## x2 0.00 1.00 0.28
```

```
## y  0.66 0.28 1.00
##
## n= 12
##
##
## P
##    x1     x2     y
## x1        1.0000 0.0193
## x2 1.0000        0.3718
## y  0.0193 0.3718
```

2.1.2 Fit second-order model

Because this is a special kind of model (a full second-order model), we can get the test for higher order terms and lack of fit simply by using `rsm()`.

Fit second-order linear model.

```
# load the rsm package
library(rsm)

# fit second-order (SO) model
#   -- look up ?SO and see other options
rsm.2.8.y.SOx12 <- rsm(y ~ SO(x1, x2), data = df.2.8)

# which variables are available in the rsm object?
names(rsm.2.8.y.SOx12)

##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
## [13] "data"          "b"             "order"         "B"
## [17] "newlabs"

# which variables are available in the summary of the rsm object?
names(summary(rsm.2.8.y.SOx12))

##  [1] "call"          "terms"         "residuals"     "coefficients"
##  [5] "aliased"       "sigma"         "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"  "canonical"
## [13] "lof"

# show the summary
summary(rsm.2.8.y.SOx12)

##
## Call:
## rsm(formula = y ~ SO(x1, x2), data = df.2.8)
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   79.750      1.214   65.71  8.4e-10 ***
## x1             9.825      0.858   11.45  2.7e-05 ***
## x2             4.216      0.858    4.91  0.00268 **
## x1:x2         -7.750      1.214   -6.39  0.00069 ***
```

```
## x1^2          -8.875      0.959   -9.25  9.0e-05 ***
## x2^2          -5.125      0.959   -5.34  0.00176 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.98,Adjusted R-squared:  0.963
## F-statistic: 58.9 on 5 and 6 DF,  p-value: 5.12e-05
##
## Analysis of Variance Table
##
## Response: y
##              Df Sum Sq Mean Sq F value  Pr(>F)
## FO(x1, x2)   2    914     457   77.61 5.2e-05
## TWI(x1, x2)  1    240     240   40.78 0.00069
## PQ(x1, x2)   2    579     289   49.13 0.00019
## Residuals    6     35       6
## Lack of fit  3      9       3    0.32 0.81192
## Pure error   3     27       9
##
## Stationary point of response surface:
##       x1       x2
##  0.55819 -0.01073
##
## Eigenanalysis:
## $values
## [1]  -2.695 -11.305
##
## $vectors
##        [,1]    [,2]
## x1  0.5312 -0.8472
## x2 -0.8472 -0.5312
# include externally Studentized residuals in the rsm object for plotting later
rsm.2.8.y.SOx12$studres <- rstudent(rsm.2.8.y.SOx12)
```

2.1.3 Model fitting

The following illustrates fitting several model types using `rsm()`.

Fit a first-order model.

```
# fit the first-order model
rsm.2.8.y.FOx12 <- rsm(y ~ FO(x1, x2), data = df.2.8)
# externally Studentized residuals
rsm.2.8.y.FOx12$studres <- rstudent(rsm.2.8.y.FOx12)
summary(rsm.2.8.y.FOx12)
##
## Call:
## rsm(formula = y ~ FO(x1, x2), data = df.2.8)
##
##              Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)    70.42       2.81   25.03  1.2e-09 ***
## x1              9.82       3.45    2.85    0.019 *
## x2              4.22       3.45    1.22    0.252
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.517,Adjusted R-squared:  0.41
## F-statistic: 4.82 on 2 and 9 DF,  p-value: 0.0378
##
## Analysis of Variance Table
##
## Response: y
##             Df Sum Sq Mean Sq F value Pr(>F)
## FO(x1, x2)   2    914     457    4.82  0.038
## Residuals    9    855      95
## Lack of fit  6    828     138   15.47  0.023
## Pure error   3     27       9
##
## Direction of steepest ascent (at radius 1):
##     x1     x2
## 0.9190 0.3943
##
## Corresponding increment in original units:
##     x1     x2
## 0.9190 0.3943
```

Fit a first-order with two-way interaction model

```
# fit the first-order with two-way interaction model.
rsm.2.8.y.TWIx12 <- rsm(y ~ FO(x1, x2) + TWI(x1, x2), data = df.2.8)
# externally Studentized residuals
rsm.2.8.y.TWIx12$studres <- rstudent(rsm.2.8.y.TWIx12)
summary(rsm.2.8.y.TWIx12)
```

```
##
## Call:
## rsm(formula = y ~ FO(x1, x2) + TWI(x1, x2), data = df.2.8)
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    70.42       2.53   27.84    3e-09 ***
## x1              9.82       3.10    3.17    0.013 *
## x2              4.22       3.10    1.36    0.211
## x1:x2          -7.75       4.38   -1.77    0.115
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.653,Adjusted R-squared:  0.523
## F-statistic: 5.01 on 3 and 8 DF,  p-value: 0.0304
##
## Analysis of Variance Table
##
## Response: y
##             Df Sum Sq Mean Sq F value Pr(>F)
## FO(x1, x2)   2    914     457    5.95  0.026
```

```
## TWI(x1, x2)  1    240     240    3.13  0.115
## Residuals    8    614      77
## Lack of fit  5    588     118   13.18  0.030
## Pure error   3     27       9
##
## Stationary point of response surface:
##    x1    x2
## 0.544 1.268
##
## Eigenanalysis:
## $values
## [1]  3.875 -3.875
##
## $vectors
##        [,1]    [,2]
## x1 -0.7071 -0.7071
## x2  0.7071 -0.7071
```

Fit a second-order without interactions model.

```
# Fit the second-order without interactions model
rsm.2.8.y.PQx12 <- rsm(y ~ FO(x1, x2) + PQ(x1, x2), data = df.2.8)
# externally Studentized residuals
rsm.2.8.y.PQx12$studres <- rstudent(rsm.2.8.y.PQx12)
summary(rsm.2.8.y.PQx12)
```

```
##
## Call:
## rsm(formula = y ~ FO(x1, x2) + PQ(x1, x2), data = df.2.8)
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    79.75       3.14   25.42  3.7e-08 ***
## x1              9.82       2.22    4.43    0.003 **
## x2              4.22       2.22    1.90    0.099 .
## x1^2           -8.87       2.48   -3.58    0.009 **
## x2^2           -5.12       2.48   -2.07    0.078 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.844,Adjusted R-squared:  0.755
## F-statistic: 9.48 on 4 and 7 DF,  p-value: 0.0059
##
## Analysis of Variance Table
##
## Response: y
##             Df Sum Sq Mean Sq F value Pr(>F)
## FO(x1, x2)   2    914     457   11.61  0.006
## PQ(x1, x2)   2    579     289    7.35  0.019
## Residuals    7    276      39
## Lack of fit  4    249      62    6.98  0.071
## Pure error   3     27       9
##
## Stationary point of response surface:
##     x1     x2
```

```
## 0.5535 0.4113
##
## Eigenanalysis:
## $values
## [1] -5.125 -8.875
##
## $vectors
##    [,1] [,2]
## x1    0   -1
## x2   -1    0
```

```
## There are at least two ways of specifying the full second-order model
#  SO(x1, x2) = FO(x1, x2) + TWI(x1, x2) + PQ(x1, x2)
#              = x1 + x2 + x1:x2 + x1^2 + x2^2
#              = (x1 + x2)^2
```

2.1.4 Model comparisons

Model comparison (for multi-parameter tests).

```
# compare the reduced first-order model to the full second-order model
anova(rsm.2.8.y.FOx12, rsm.2.8.y.SOx12)
```

```
## Analysis of Variance Table
##
## Model 1: y ~ FO(x1, x2)
## Model 2: y ~ FO(x1, x2) + TWI(x1, x2) + PQ(x1, x2)
##   Res.Df RSS Df Sum of Sq    F  Pr(>F)
## 1      9 855
## 2      6  35  3       819 46.4 0.00015 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Confidence intervals and prediction intervals.

```
# conf int for parameters
confint(rsm.2.8.y.SOx12)
```

```
##                    2.5 % 97.5 %
## (Intercept)       76.780 82.720
## FO(x1, x2)x1       7.725 11.925
## FO(x1, x2)x2       2.116  6.316
## TWI(x1, x2)      -10.720 -4.780
## PQ(x1, x2)x1^2   -11.223 -6.527
## PQ(x1, x2)x2^2    -7.473 -2.777
```

```
# conf int for regression line
predict(rsm.2.8.y.SOx12, df.2.8[1:dim(df.2.8)[1],], interval = "confidence")
```

```
##     fit   lwr   upr
## 1 43.96 39.26 48.65
## 2 79.11 74.41 83.80
## 3 67.89 63.20 72.59
## 4 72.04 67.35 76.74
## 5 48.11 43.41 52.80
```

```
## 6  75.89 71.20 80.59
## 7  63.54 58.84 68.23
## 8  75.46 70.77 80.16
## 9  79.75 76.78 82.72
## 10 79.75 76.78 82.72
## 11 79.75 76.78 82.72
## 12 79.75 76.78 82.72
```

```
# pred int for new observations
predict(rsm.2.8.y.SOx12, df.2.8[1:dim(df.2.8)[1],], interval = "prediction")
```

```
##      fit   lwr   upr
## 1  43.96 36.39 51.53
## 2  79.11 71.54 86.68
## 3  67.89 60.32 75.46
## 4  72.04 64.47 79.61
## 5  48.11 40.53 55.68
## 6  75.89 68.32 83.47
## 7  63.54 55.97 71.11
## 8  75.46 67.89 83.03
## 9  79.75 73.11 86.39
## 10 79.75 73.11 86.39
## 11 79.75 73.11 86.39
## 12 79.75 73.11 86.39
```

2.1.5 Lack-of-fit test

The lack-of-fit (LOF) test is equivalent to a comparison between two models. First, we define the full model by setting up a categorical group variable that will take unique values for each distinct pair of (x1, x2) values. This group variable fits a model that is equivalent to a one-way ANOVA. The SSE for the full model is 26.75 with 3 df. This is the pure error SS and df. (Try this for yourself.) Second, we define the reduced model (reduced compared to the one-way ANOVA above) as the regression model fit (taking x1 and x2 as continuous variables). The SSE for the reduced model is 35.35 with 6 df. This is the residual SS. The LOF SS is the difference of SSE between the reduced and the full model: $35.35 - 26.75 = 8.6$ with $6 - 3 = 3$ df. The F-test is then the LOF SSE and df vs the full SSE and df, $F = (8.6/3)/(26.75/3) = 0.32$, where there are 3 df and 3 df in the numerator and denominator

```
summary(rsm.2.8.y.SOx12)$lof
```

```
## Analysis of Variance Table
##
## Response: y
##             Df Sum Sq Mean Sq F value  Pr(>F)
```

```
## FO(x1, x2)   2    914     457   77.61 5.2e-05 ***
## TWI(x1, x2)  1    240     240   40.78 0.00069 ***
## PQ(x1, x2)   2    579     289   49.13 0.00019 ***
## Residuals    6     35       6
## Lack of fit  3      9       3    0.32 0.81192
## Pure error   3     27       9
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2.1.6 Diagnostics and plots of estimated response surfaces

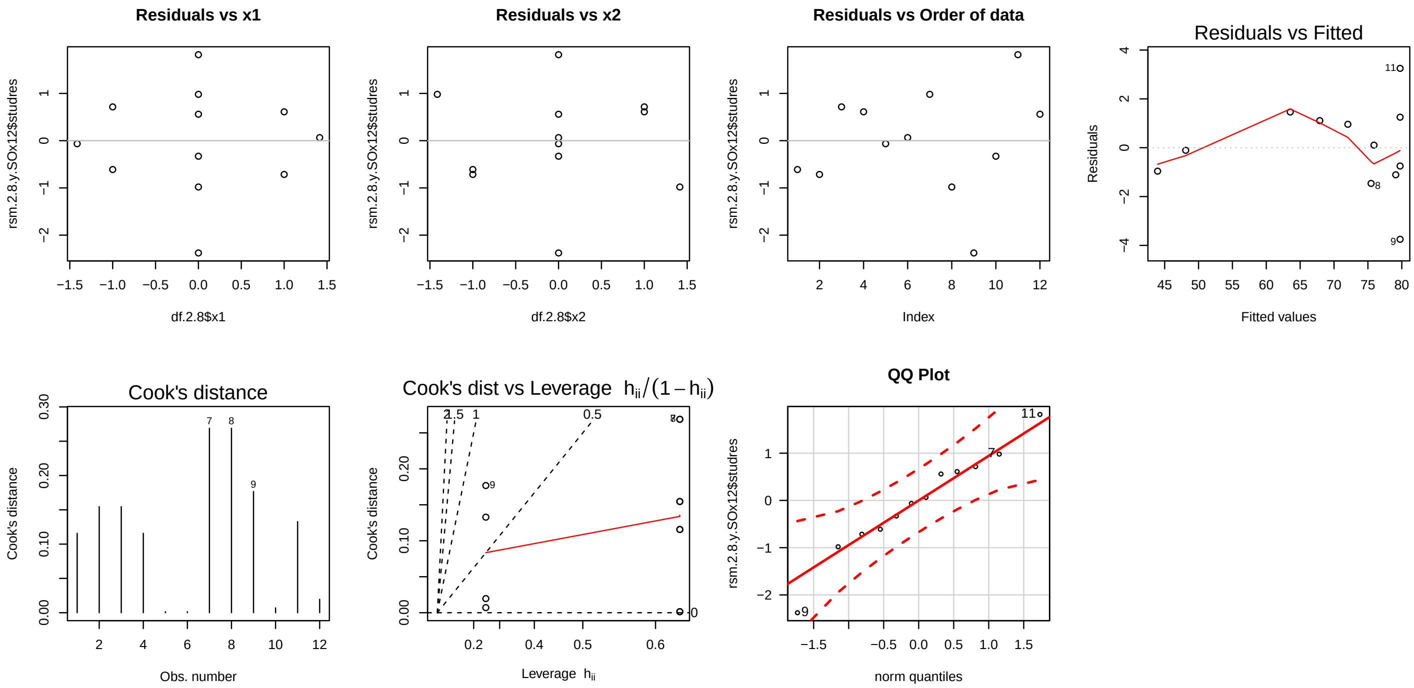
Plot the residuals, and comment on model adequacy.

```
# plot diagnostics
par(mfrow=c(2,4))

plot(df.2.8$x1, rsm.2.8.y.SOx12$studres, main="Residuals vs x1")
  # horizontal line at zero
  abline(h = 0, col = "gray75")

plot(df.2.8$x2, rsm.2.8.y.SOx12$studres, main="Residuals vs x2")
  # horizontal line at zero
  abline(h = 0, col = "gray75")

# residuals vs order of data
plot(rsm.2.8.y.SOx12$studres, main="Residuals vs Order of data")
  # horizontal line at zero
  abline(h = 0, col = "gray75")

plot(rsm.2.8.y.SOx12, which = c(1,4,6))

# Normality of Residuals
library(car)
qqPlot(rsm.2.8.y.SOx12$studres, las = 1, id.n = 3, main="QQ Plot")
```

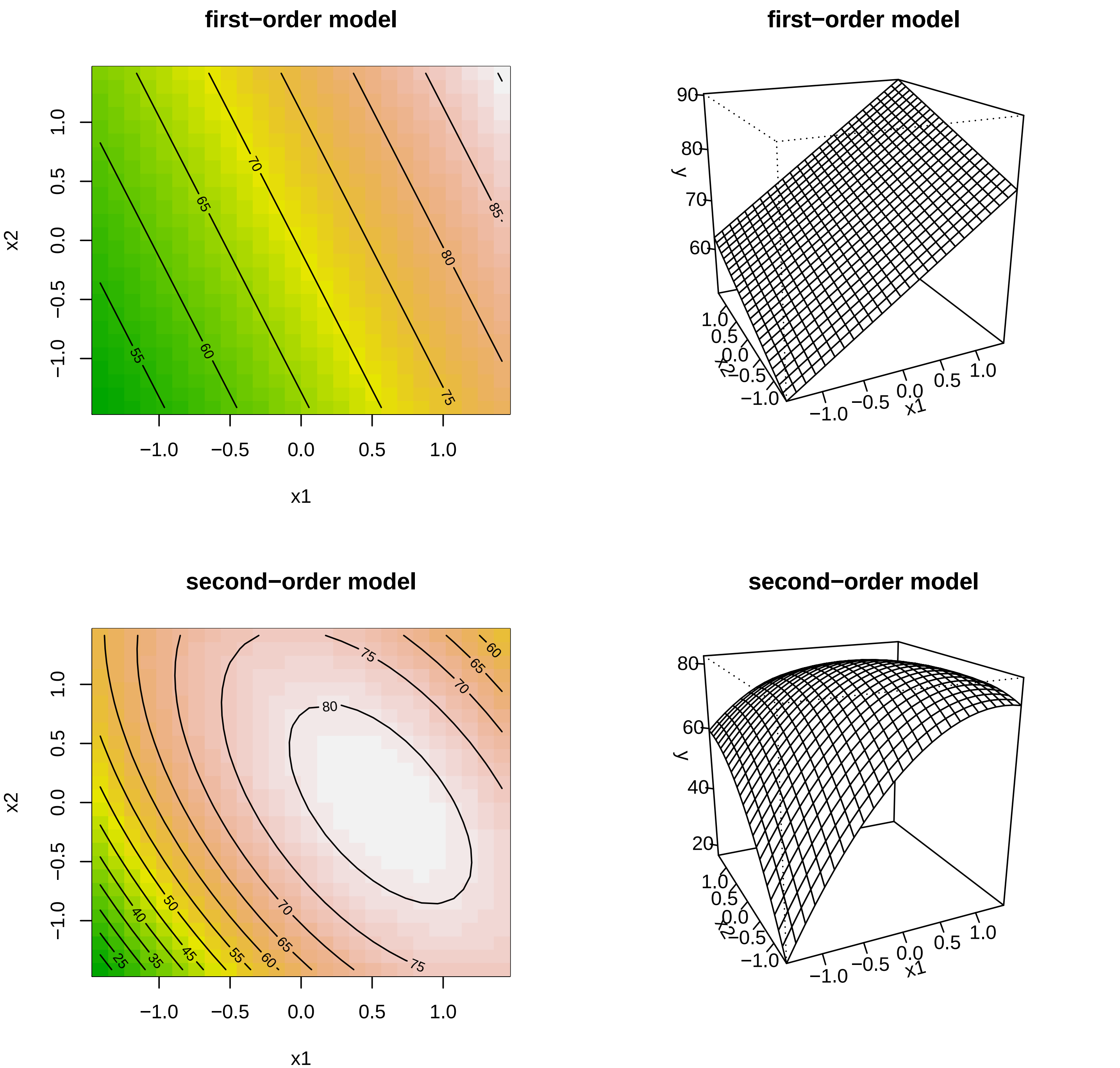
```
##  9 11  7
##  1 12 11
```

```
cooks.distance(rsm.2.8.y.SOx12)
```

```
##        1        2        3        4        5        6        7
## 0.115698 0.154570 0.154570 0.115698 0.001405 0.001405 0.268863
##        8        9       10       11       12
## 0.268863 0.176813 0.007073 0.132806 0.019646
```




```
# first-order model
par(mfrow=c(2,2))
contour(rsm.2.8.y.FOx12, ~ x1 + x2, image = TRUE, main="first-order model")
persp(rsm.2.8.y.FOx12, x2 ~ x1, zlab = "y", main="first-order model")

# second-order model
contour(rsm.2.8.y.SOx12, ~ x1 + x2, image = TRUE, main="second-order model")
persp(rsm.2.8.y.SOx12, x2 ~ x1, zlab = "y", main="second-order model")
```


first−order model
x1
x2
first−order model
y
x1
x2
second−order model
x1
x2
second−order model
y
x1
x2

Chapter 3

Two-Level Factorial Designs

3.1 Example 3.1, Table 3.5, p. 90

Read data and convert to long format.

```
#### 3.1
fn.data <- "http://statacumen.com/teach/RSM/data/RSM_EX_03-01.txt"
df.3.1 <- read.table(fn.data, header=TRUE)
str(df.3.1)
```

```
## 'data.frame': 8 obs. of  5 variables:
##  $ a : int  -1 1 -1 1 -1 1 -1 1
##  $ b : int  -1 -1 1 1 -1 -1 1 1
##  $ c : int  -1 -1 -1 -1 1 1 1 1
##  $ y1: int  247 470 429 435 837 551 775 660
##  $ y2: int  400 446 405 445 850 670 865 530
```

```
df.3.1
```

```
##    a  b  c  y1  y2
## 1 -1 -1 -1 247 400
## 2  1 -1 -1 470 446
## 3 -1  1 -1 429 405
## 4  1  1 -1 435 445
## 5 -1 -1  1 837 850
## 6  1 -1  1 551 670
## 7 -1  1  1 775 865
## 8  1  1  1 660 530
```

```
# reshape data into long format
library(reshape2)
df.3.1.long <- melt(df.3.1, id.vars = c("a", "b", "c")
                    , variable.name = "rep", value.name = "y")
df.3.1.long
```

```
##     a  b  c rep   y
## 1  -1 -1 -1  y1 247
## 2   1 -1 -1  y1 470
## 3  -1  1 -1  y1 429
## 4   1  1 -1  y1 435
## 5  -1 -1  1  y1 837
## 6   1 -1  1  y1 551
## 7  -1  1  1  y1 775
## 8   1  1  1  y1 660
## 9  -1 -1 -1  y2 400
## 10  1 -1 -1  y2 446
## 11 -1  1 -1  y2 405
## 12  1  1 -1  y2 445
## 13 -1 -1  1  y2 850
## 14  1 -1  1  y2 670
## 15 -1  1  1  y2 865
## 16  1  1  1  y2 530
```

Fit second-order linear model.

```
library(rsm)
rsm.3.1.y.SOabc <- rsm(y ~ SO(a, b, c), data = df.3.1.long)
```

```
## Warning:  Some coefficients are aliased - cannot use 'rsm' methods.
##  Returning an 'lm' object.
# externally Studentized residuals
rsm.3.1.y.SOabc$studres <- rstudent(rsm.3.1.y.SOabc)
summary(rsm.3.1.y.SOabc)
```

```
##
## Call:
## rsm(formula = y ~ SO(a, b, c), data = df.3.1.long)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -91.44 -27.47   5.69  27.72  79.94
##
## Coefficients: (3 not defined because of singularities)
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)       563.44      15.72   35.85  5.1e-11 ***
## FO(a, b, c)a      -37.56      15.72   -2.39  0.04055 *
## FO(a, b, c)b        4.56      15.72    0.29  0.77815
## FO(a, b, c)c      153.81      15.72    9.79  4.3e-06 ***
## TWI(a, b, c)a:b   -12.94      15.72   -0.82  0.43165
## TWI(a, b, c)a:c   -76.94      15.72   -4.90  0.00085 ***
## TWI(a, b, c)b:c   -14.31      15.72   -0.91  0.38619
## PQ(a, b, c)a^2        NA         NA      NA       NA
## PQ(a, b, c)b^2        NA         NA      NA       NA
## PQ(a, b, c)c^2        NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 62.9 on 9 degrees of freedom
## Multiple R-squared:  0.934,Adjusted R-squared:  0.89
## F-statistic: 21.2 on 6 and 9 DF,  p-value: 7.88e-05
```

Note that the quadratic terms can't be estimated because there are only two levels for each predictor variable.

Fit main-effect with three-way interaction linear model.

```
lm.3.1.y.3WIabc <- lm(y ~ (a + b + c)^3, data = df.3.1.long)
# externally Studentized residuals
lm.3.1.y.3WIabc$studres <- rstudent(lm.3.1.y.3WIabc)
summary(lm.3.1.y.3WIabc)
```

```
##
## Call:
## lm.default(formula = y ~ (a + b + c)^3, data = df.3.1.long)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -76.5  -20.2    0.0   20.2   76.5
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   563.44      15.81   35.64  4.2e-10 ***
```

```
## a            -37.56      15.81   -2.38   0.0448 *
## b              4.56      15.81    0.29   0.7802
## c            153.81      15.81    9.73  1.0e-05 ***
## a:b          -12.94      15.81   -0.82   0.4369
## a:c          -76.94      15.81   -4.87   0.0012 **
## b:c          -14.31      15.81   -0.91   0.3918
## a:b:c         14.94      15.81    0.94   0.3724
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 63.2 on 8 degrees of freedom
## Multiple R-squared:  0.94,Adjusted R-squared:  0.888
## F-statistic: 18.1 on 7 and 8 DF,  p-value: 0.00026
```

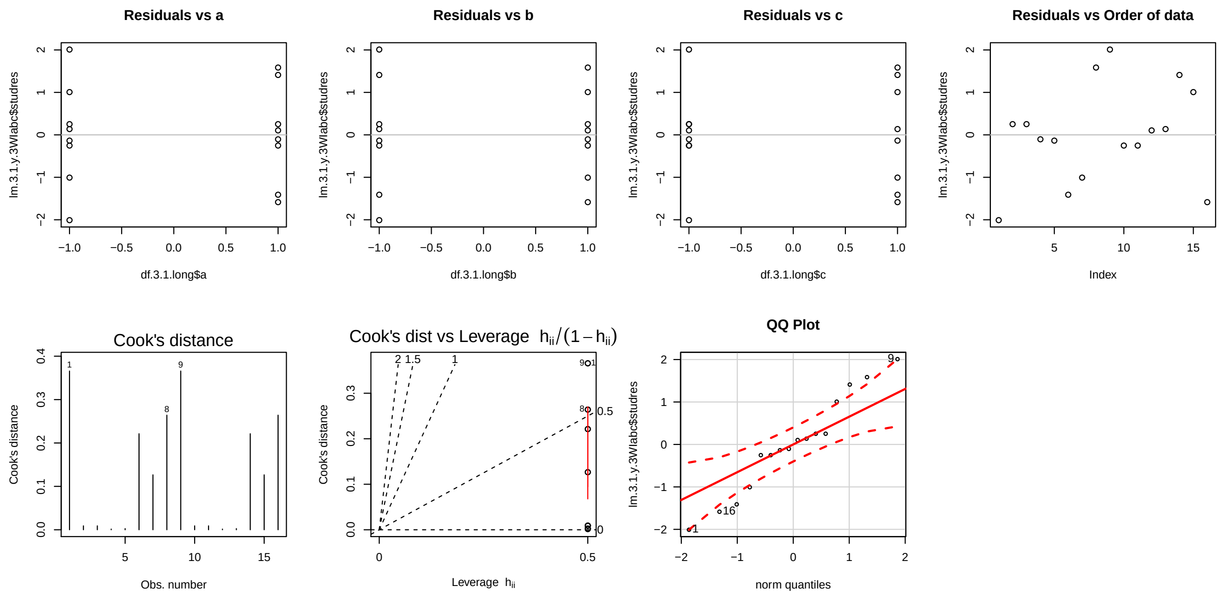
Here are the residual diagnostic plots, but we'll skip over them since our interest is in the interaction plots below.

```
# plot diagnostics
par(mfrow=c(2,4))

plot(df.3.1.long$a, lm.3.1.y.3WIabc$studres, main="Residuals vs a")
  # horizontal line at zero
  abline(h = 0, col = "gray75")

plot(df.3.1.long$b, lm.3.1.y.3WIabc$studres, main="Residuals vs b")
  # horizontal line at zero
  abline(h = 0, col = "gray75")

plot(df.3.1.long$c, lm.3.1.y.3WIabc$studres, main="Residuals vs c")
  # horizontal line at zero
  abline(h = 0, col = "gray75")

# residuals vs order of data
plot(lm.3.1.y.3WIabc$studres, main="Residuals vs Order of data")
  # horizontal line at zero
  abline(h = 0, col = "gray75")

plot(lm.3.1.y.3WIabc, which = c(4,6))

# Normality of Residuals
library(car)
qqPlot(lm.3.1.y.3WIabc$studres, las = 1, id.n = 3, main="QQ Plot")
```

```
##  1  9 16
##  1 16  2
```

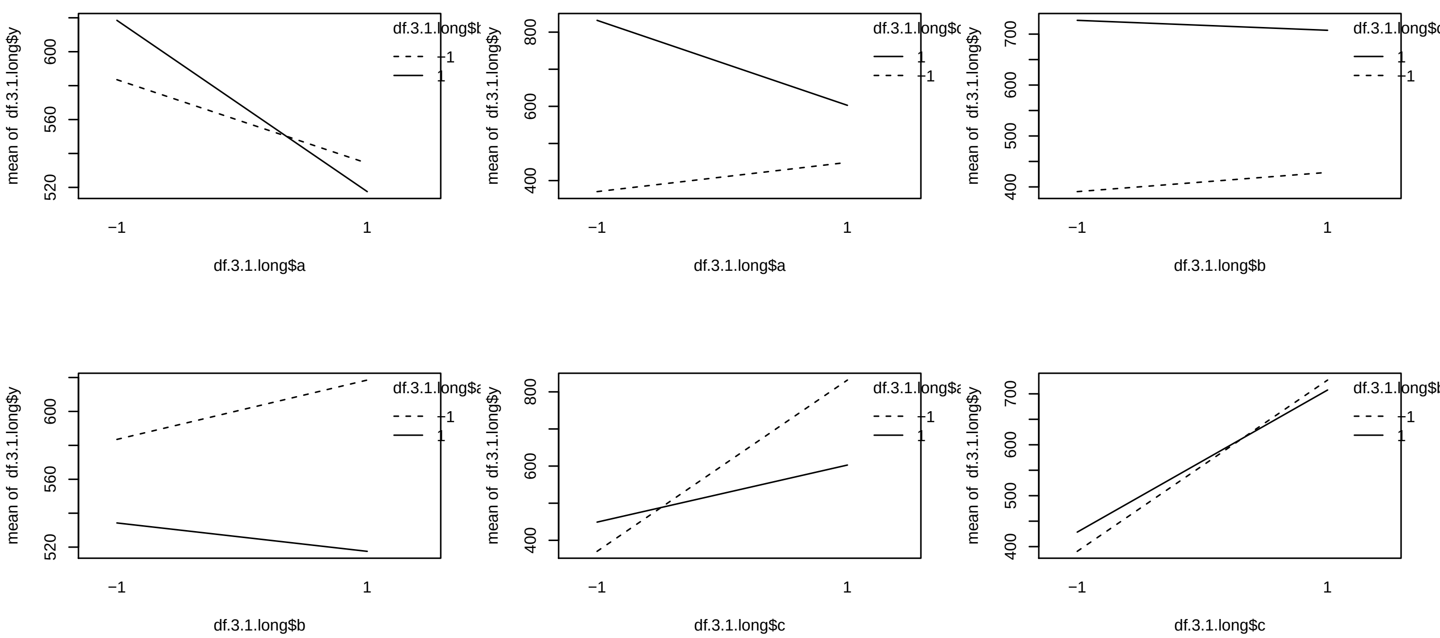
```
cooks.distance(lm.3.1.y.3WIabc)
```

```
##        1        2        3        4        5        6        7
## 0.365817 0.009001 0.009001 0.001563 0.002641 0.221297 0.126580
##        8        9       10       11       12       13       14
## 0.264100 0.365817 0.009001 0.009001 0.001563 0.002641 0.221297
##       15       16
## 0.126580 0.264100
```



Interaction plots, main effects vs main effects.

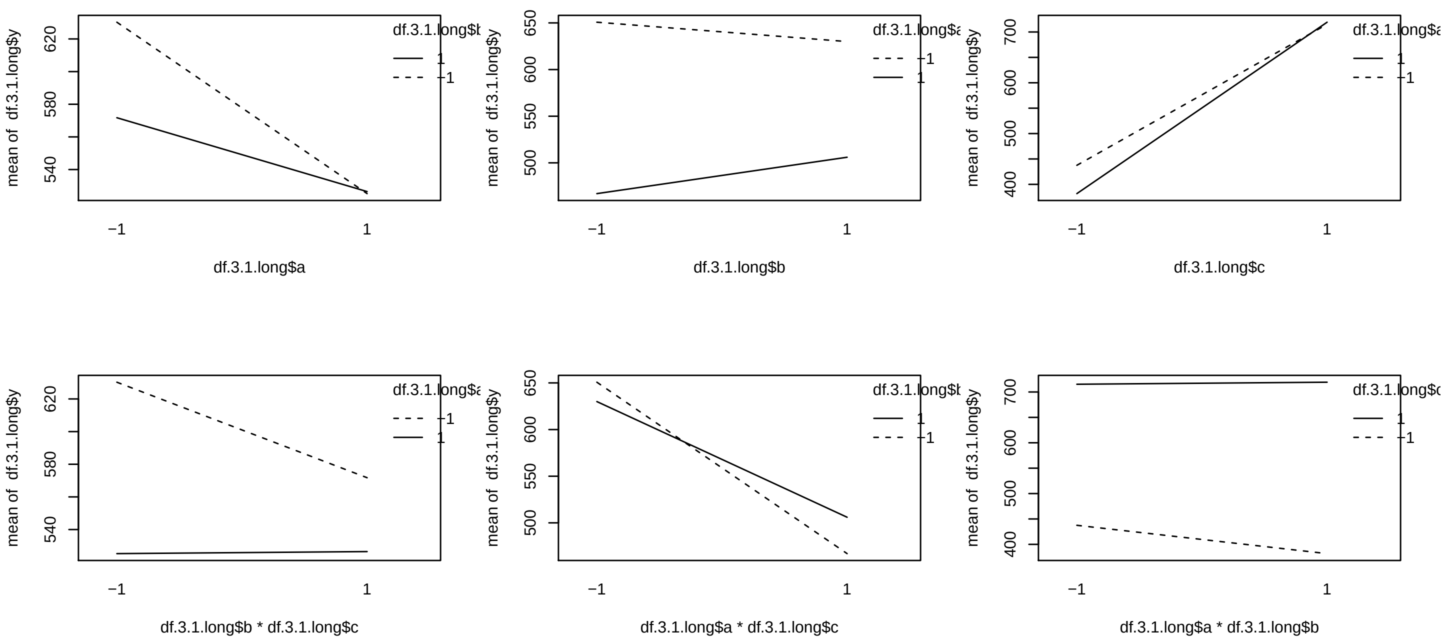
```
# interaction plot
par(mfcol=c(2,3))
interaction.plot(df.3.1.long$a, df.3.1.long$b, df.3.1.long$y)
interaction.plot(df.3.1.long$b, df.3.1.long$a, df.3.1.long$y)
interaction.plot(df.3.1.long$a, df.3.1.long$c, df.3.1.long$y)
interaction.plot(df.3.1.long$c, df.3.1.long$a, df.3.1.long$y)
interaction.plot(df.3.1.long$b, df.3.1.long$c, df.3.1.long$y)
interaction.plot(df.3.1.long$c, df.3.1.long$b, df.3.1.long$y)
```



Interaction plots, main effects vs two-way interactions.

```
# interaction plot
par(mfcol=c(2,3))
interaction.plot(df.3.1.long$a, df.3.1.long$b * df.3.1.long$c, df.3.1.long$y)
interaction.plot(df.3.1.long$b * df.3.1.long$c, df.3.1.long$a, df.3.1.long$y)
interaction.plot(df.3.1.long$b, df.3.1.long$a * df.3.1.long$c, df.3.1.long$y)
```

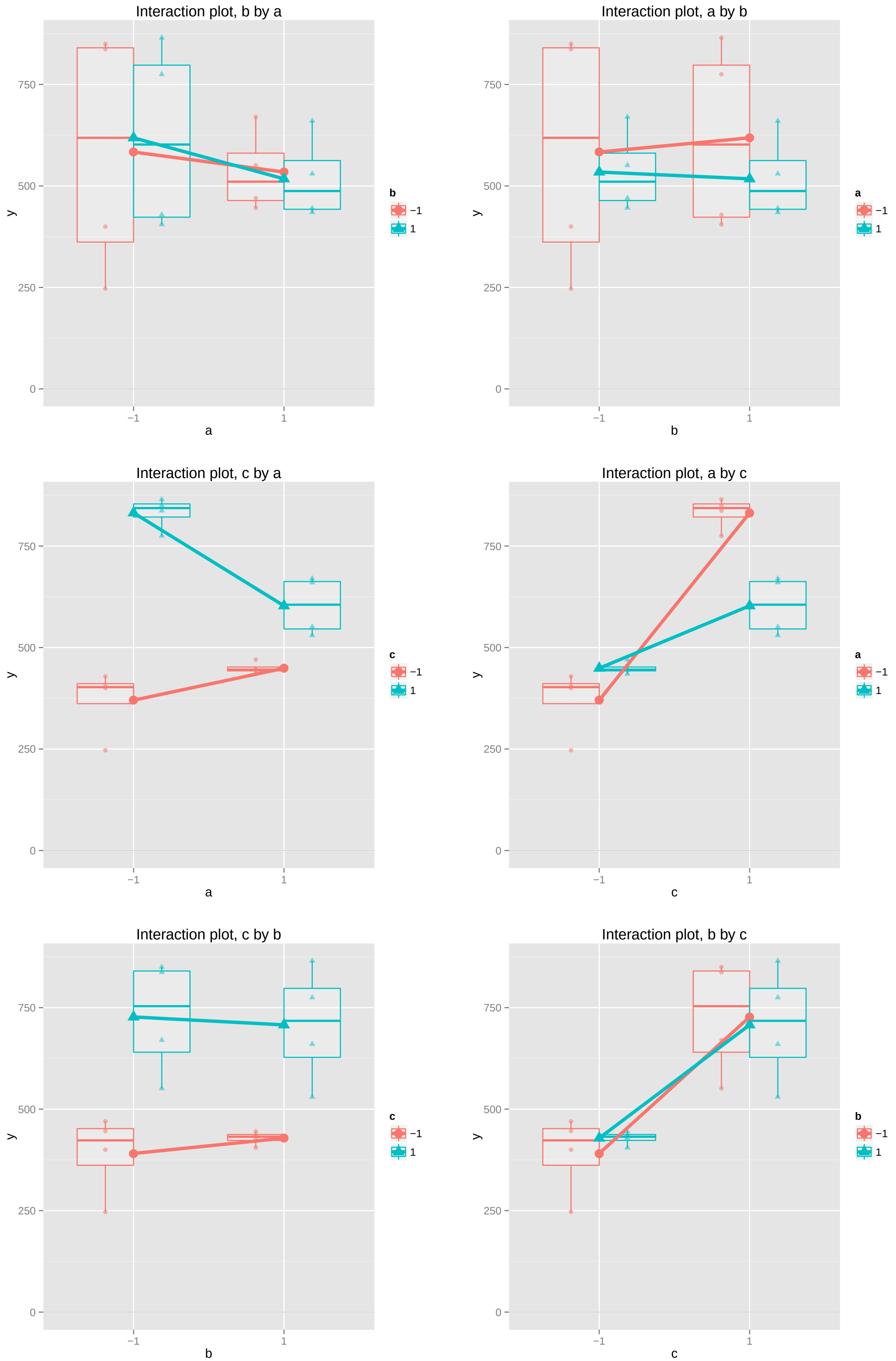
```
interaction.plot(df.3.1.long$a * df.3.1.long$c, df.3.1.long$b, df.3.1.long$y)
interaction.plot(df.3.1.long$c, df.3.1.long$a * df.3.1.long$b, df.3.1.long$y)
interaction.plot(df.3.1.long$a * df.3.1.long$b, df.3.1.long$c, df.3.1.long$y)
```



Interaction plots, main effects vs main effects in `ggplot()`.

```
# Interaction plots, ggplot
library(plyr)
# Calculate the cell means for each (a, b) combination
# create factor version for ggplot categories
df.3.1.factor <- df.3.1.long
df.3.1.factor$a <- factor(df.3.1.factor$a)
df.3.1.factor$b <- factor(df.3.1.factor$b)
df.3.1.factor$c <- factor(df.3.1.factor$c)

#mean(df.3.1.factor[, "y"])
df.3.1.factor.mean    <- ddply(df.3.1.factor, .(), summarise, m = mean(y))
#df.3.1.factor.mean
df.3.1.factor.mean.a  <- ddply(df.3.1.factor, .(a), summarise, m = mean(y))
#df.3.1.factor.mean.a
df.3.1.factor.mean.b  <- ddply(df.3.1.factor, .(b), summarise, m = mean(y))
#df.3.1.factor.mean.b
df.3.1.factor.mean.c  <- ddply(df.3.1.factor, .(c), summarise, m = mean(y))
#df.3.1.factor.mean.c
df.3.1.factor.mean.ab <- ddply(df.3.1.factor, .(a,b), summarise, m = mean(y))
#df.3.1.factor.mean.ab
df.3.1.factor.mean.ac <- ddply(df.3.1.factor, .(a,c), summarise, m = mean(y))
#df.3.1.factor.mean.ac
df.3.1.factor.mean.bc <- ddply(df.3.1.factor, .(b,c), summarise, m = mean(y))
#df.3.1.factor.mean.bc
df.3.1.factor.mean.abc <- ddply(df.3.1.factor, .(a,b,c), summarise, m = mean(y))
#df.3.1.factor.mean.abc

library(ggplot2)
## (a, b)
p <- ggplot(df.3.1.factor, aes(x = a, y = y, colour = b, shape = b))
p <- p + geom_hline(aes(yintercept = 0), colour = "black"
                , linetype = "solid", size = 0.2, alpha = 0.3)
p <- p + geom_boxplot(alpha = 0.25, outlier.size=0.1)
p <- p + geom_point(alpha = 0.5, position=position_dodge(width=0.75))
p <- p + geom_point(data = df.3.1.factor.mean.ab, aes(y = m), size = 4)
```

```
p <- p + geom_line(data = df.3.1.factor.mean.ab, aes(y = m, group = b), size = 1.5)
p <- p + labs(title = "Interaction plot, b by a")
print(p)
## ymax not defined:  adjusting position using y instead
p <- ggplot(df.3.1.factor, aes(x = b, y = y, colour = a, shape = a))
p <- p + geom_hline(aes(yintercept = 0), colour = "black"
                  , linetype = "solid", size = 0.2, alpha = 0.3)
p <- p + geom_boxplot(alpha = 0.25, outlier.size=0.1)
p <- p + geom_point(alpha = 0.5, position=position_dodge(width=0.75))
p <- p + geom_point(data = df.3.1.factor.mean.ab, aes(y = m), size = 4)
p <- p + geom_line(data = df.3.1.factor.mean.ab, aes(y = m, group = a), size = 1.5)
p <- p + labs(title = "Interaction plot, a by b")
print(p)
## ymax not defined:  adjusting position using y instead
## (a, c)
p <- ggplot(df.3.1.factor, aes(x = a, y = y, colour = c, shape = c))
p <- p + geom_hline(aes(yintercept = 0), colour = "black"
                  , linetype = "solid", size = 0.2, alpha = 0.3)
p <- p + geom_boxplot(alpha = 0.25, outlier.size=0.1)
p <- p + geom_point(alpha = 0.5, position=position_dodge(width=0.75))
p <- p + geom_point(data = df.3.1.factor.mean.ac, aes(y = m), size = 4)
p <- p + geom_line(data = df.3.1.factor.mean.ac, aes(y = m, group = c), size = 1.5)
p <- p + labs(title = "Interaction plot, c by a")
print(p)
## ymax not defined:  adjusting position using y instead
p <- ggplot(df.3.1.factor, aes(x = c, y = y, colour = a, shape = a))
p <- p + geom_hline(aes(yintercept = 0), colour = "black"
                  , linetype = "solid", size = 0.2, alpha = 0.3)
p <- p + geom_boxplot(alpha = 0.25, outlier.size=0.1)
p <- p + geom_point(alpha = 0.5, position=position_dodge(width=0.75))
p <- p + geom_point(data = df.3.1.factor.mean.ac, aes(y = m), size = 4)
p <- p + geom_line(data = df.3.1.factor.mean.ac, aes(y = m, group = a), size = 1.5)
p <- p + labs(title = "Interaction plot, a by c")
print(p)
## ymax not defined:  adjusting position using y instead
## (b, c)
p <- ggplot(df.3.1.factor, aes(x = b, y = y, colour = c, shape = c))
p <- p + geom_hline(aes(yintercept = 0), colour = "black"
                  , linetype = "solid", size = 0.2, alpha = 0.3)
p <- p + geom_boxplot(alpha = 0.25, outlier.size=0.1)
p <- p + geom_point(alpha = 0.5, position=position_dodge(width=0.75))
p <- p + geom_point(data = df.3.1.factor.mean.bc, aes(y = m), size = 4)
p <- p + geom_line(data = df.3.1.factor.mean.bc, aes(y = m, group = c), size = 1.5)
p <- p + labs(title = "Interaction plot, c by b")
print(p)
## ymax not defined:  adjusting position using y instead
p <- ggplot(df.3.1.factor, aes(x = c, y = y, colour = b, shape = b))
p <- p + geom_hline(aes(yintercept = 0), colour = "black"
                  , linetype = "solid", size = 0.2, alpha = 0.3)
p <- p + geom_boxplot(alpha = 0.25, outlier.size=0.1)
p <- p + geom_point(alpha = 0.5, position=position_dodge(width=0.75))
p <- p + geom_point(data = df.3.1.factor.mean.bc, aes(y = m), size = 4)
p <- p + geom_line(data = df.3.1.factor.mean.bc, aes(y = m, group = b), size = 1.5)
p <- p + labs(title = "Interaction plot, b by c")
print(p)
## ymax not defined:  adjusting position using y instead
```


Interaction plot, b by a
Interaction plot, a by b
Interaction plot, c by a
Interaction plot, a by c
Interaction plot, c by b
Interaction plot, b by c
y
0
250
500
750
−1
1
a
b
c
−1
1

3.2 Example 3.2, Table 3.7, p. 97

Here's a quick way to create a design matrix for a factorial design.

```
design <- expand.grid("a" = c(-1, 1)
                    , "b" = c(-1, 1)
                    , "c" = c(-1, 1)
                    , "d" = c(-1, 1)
                    , "y" = NA)
design
```

```
##     a  b  c  d  y
## 1  -1 -1 -1 -1 NA
## 2   1 -1 -1 -1 NA
## 3  -1  1 -1 -1 NA
## 4   1  1 -1 -1 NA
## 5  -1 -1  1 -1 NA
## 6   1 -1  1 -1 NA
## 7  -1  1  1 -1 NA
## 8   1  1  1 -1 NA
## 9  -1 -1 -1  1 NA
## 10  1 -1 -1  1 NA
## 11 -1  1 -1  1 NA
## 12  1  1 -1  1 NA
## 13 -1 -1  1  1 NA
## 14  1 -1  1  1 NA
## 15 -1  1  1  1 NA
## 16  1  1  1  1 NA
```

Read data.

```
#### 3.2
fn.data <- "http://statacumen.com/teach/RSM/data/RSM_EX_03-02.txt"
df.3.2 <- read.table(fn.data, header=TRUE)
str(df.3.2)
```

```
## 'data.frame': 16 obs. of  5 variables:
##  $ a: int  -1 1 -1 1 -1 1 -1 1 -1 1 ...
##  $ b: int  -1 -1 1 1 -1 -1 1 1 -1 -1 ...
##  $ c: int  -1 -1 -1 -1 1 1 1 1 -1 -1 ...
##  $ d: int  -1 -1 -1 -1 -1 -1 -1 -1 1 1 ...
##  $ y: int  45 71 48 65 68 60 80 65 43 100 ...
```

```
df.3.2
```

```
##     a  b  c  d   y
## 1  -1 -1 -1 -1  45
## 2   1 -1 -1 -1  71
## 3  -1  1 -1 -1  48
## 4   1  1 -1 -1  65
## 5  -1 -1  1 -1  68
## 6   1 -1  1 -1  60
## 7  -1  1  1 -1  80
## 8   1  1  1 -1  65
## 9  -1 -1 -1  1  43
## 10  1 -1 -1  1 100
```

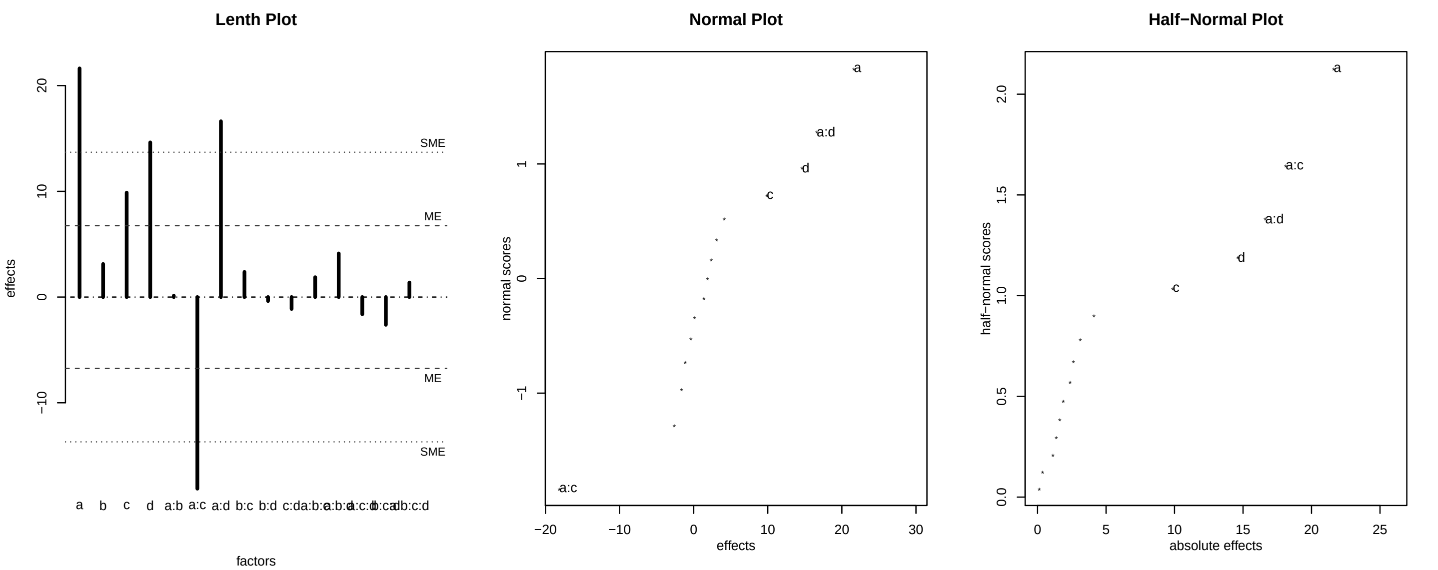
```
## 11 -1  1 -1  1  45
## 12  1  1 -1  1 104
## 13 -1 -1  1  1  75
## 14  1 -1  1  1  86
## 15 -1  1  1  1  70
## 16  1  1  1  1  96
```

Fit first-order with four-way interaction linear model.

```
lm.3.2.y.4WIabcd <- lm(y ~ (a + b + c + d)^4, data = df.3.2)
## externally Studentized residuals
#lm.3.2.y.4WIabcd$studres <- rstudent(lm.3.2.y.4WIabcd)
#summary(lm.3.2.y.4WIabcd)
```

Use Lenth procedure and normal plots to assess effects in unreplicated design.

```
# BsMD package has unreplicated factorial tests (Daniel plots (aka normal), and Lenth)
library(BsMD)
par(mfrow=c(1,3))
LenthPlot(lm.3.2.y.4WIabcd, alpha = 0.05, main = "Lenth Plot") # , adj = 0.2

##   alpha    PSE     ME    SME
##   0.050  2.625  6.748 13.699

DanielPlot(lm.3.2.y.4WIabcd, main = "Normal Plot")
DanielPlot(lm.3.2.y.4WIabcd, half = TRUE, main = "Half-Normal Plot")
```



In example 3.2, all terms with B were not significant. So, project design to a 2^3 in factors A, C, and D (exploritory, not confirmatory). Now we have two replicates in each term (A, C, D), so now have error terms. So can do ANOVA, etc. Then, we can run additional confirmatory experiments.

Fit first-order with three-way interaction linear model.

```
lm.3.2.y.4WIacd <- lm(y ~ (a + c + d)^3, data = df.3.2)
# externally Studentized residuals
lm.3.2.y.4WIacd$studres <- rstudent(lm.3.2.y.4WIacd)
summary(lm.3.2.y.4WIacd)
```

```
##
## Call:
## lm.default(formula = y ~ (a + c + d)^3, data = df.3.2)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##   -6.0   -2.5    0.0    2.5    6.0
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   70.062      1.184   59.16  7.4e-12 ***
## a             10.812      1.184    9.13  1.7e-05 ***
## c              4.938      1.184    4.17  0.00312 **
## d              7.313      1.184    6.18  0.00027 ***
## a:c           -9.063      1.184   -7.65  6.0e-05 ***
## a:d            8.312      1.184    7.02  0.00011 ***
## c:d           -0.563      1.184   -0.48  0.64748
## a:c:d         -0.813      1.184   -0.69  0.51203
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.74 on 8 degrees of freedom
## Multiple R-squared:  0.969,Adjusted R-squared:  0.941
## F-statistic: 35.3 on 7 and 8 DF,  p-value: 2.12e-05
```

3.3 Creating Fractional Factorial designs with blocks and aliasing

The `rsm` package can generage blocked designs.

```
library(rsm)

# help for cube, see examples
?cube

# create the first block
block1 <- cube( basis = ~ x1 + x2 + x3 + x4
              , n0 = 0
              , blockgen = ~ c(x1 * x2 * x3, x1 * x3 * x4)
              , randomize = FALSE
              , bid = 1)
block1
```

```
##    run.order std.order x1.as.is x2.as.is x3.as.is x4.as.is
## 1          1         1       -1       -1       -1       -1
## 6          2         2        1       -1        1       -1
## 12         3         3        1        1       -1        1
## 15         4         4       -1        1        1        1
##
## Data are stored in coded form using these coding formulas ...
## x1 ~ x1.as.is
## x2 ~ x2.as.is
## x3 ~ x3.as.is
## x4 ~ x4.as.is
```

```
# populate a list of all the blocks and print them all
block <- list()
for (i.block in 1:4) {
  block[[i.block]] <- cube( basis = ~ x1 + x2 + x3 + x4
                          , n0 = 0
                          , blockgen = ~ c(x1 * x2 * x3, x1 * x3 * x4)
                          , randomize = FALSE
                          , bid = i.block)
}
block
```

```
## [[1]]
##    run.order std.order x1.as.is x2.as.is x3.as.is x4.as.is
## 1          1         1       -1       -1       -1       -1
## 6          2         2        1       -1        1       -1
## 12         3         3        1        1       -1        1
## 15         4         4       -1        1        1        1
##
## Data are stored in coded form using these coding formulas ...
## x1 ~ x1.as.is
## x2 ~ x2.as.is
## x3 ~ x3.as.is
## x4 ~ x4.as.is
```

```
##
## [[2]]
##    run.order std.order x1.as.is x2.as.is x3.as.is x4.as.is
## 3          1         1       -1        1       -1       -1
## 8          2         2        1        1        1       -1
## 10         3         3        1       -1       -1        1
## 13         4         4       -1       -1        1        1
##
## Data are stored in coded form using these coding formulas ...
## x1 ~ x1.as.is
## x2 ~ x2.as.is
## x3 ~ x3.as.is
## x4 ~ x4.as.is
##
## [[3]]
##    run.order std.order x1.as.is x2.as.is x3.as.is x4.as.is
## 4          1         1        1        1       -1       -1
## 7          2         2       -1        1        1       -1
## 9          3         3       -1       -1       -1        1
## 14         4         4        1       -1        1        1
##
## Data are stored in coded form using these coding formulas ...
## x1 ~ x1.as.is
## x2 ~ x2.as.is
## x3 ~ x3.as.is
## x4 ~ x4.as.is
##
## [[4]]
##    run.order std.order x1.as.is x2.as.is x3.as.is x4.as.is
## 2          1         1        1       -1       -1       -1
## 5          2         2       -1       -1        1       -1
## 11         3         3       -1        1       -1        1
## 16         4         4        1        1        1        1
##
## Data are stored in coded form using these coding formulas ...
## x1 ~ x1.as.is
## x2 ~ x2.as.is
## x3 ~ x3.as.is
## x4 ~ x4.as.is
```

Alternatively, package **FrF2** is the most comprehensive R package for creating fractional factorial 2-level designs, as well as Plackett-Burman type screening designs. Regular fractional factorials default to maximum resolution minimum aberration designs and can be customized in various ways. It can also show the alias structure for regular fractional factorials of 2-level factors.

```
library(FrF2)

k = 4  # number of factors, defaults to names = A, B, ...
```

```
FrF2( nruns = 2^k
    , nfactors = k
    , blocks = 4
    , alias.block.2fis=TRUE
    , randomize = FALSE
    )
```

```
##   run.no run.no.std.rp Blocks  A  B  C  D
## 1      1         1.1.1      1 -1 -1 -1 -1
## 2      2         4.1.2      1 -1 -1  1  1
## 3      3        14.1.3      1  1  1 -1  1
## 4      4        15.1.4      1  1  1  1 -1
##   run.no run.no.std.rp Blocks  A  B  C  D
## 5      5         5.2.1      2 -1  1 -1 -1
## 6      6         8.2.2      2 -1  1  1  1
## 7      7        10.2.3      2  1 -1 -1  1
## 8      8        11.2.4      2  1 -1  1 -1
##    run.no run.no.std.rp Blocks  A  B  C  D
## 9       9         6.3.1      3 -1  1 -1  1
## 10     10         7.3.2      3 -1  1  1 -1
## 11     11         9.3.3      3  1 -1 -1 -1
## 12     12        12.3.4      3  1 -1  1  1
##    run.no run.no.std.rp Blocks  A  B  C  D
## 13     13         2.4.1      4 -1 -1 -1  1
## 14     14         3.4.2      4 -1 -1  1 -1
## 15     15        13.4.3      4  1  1 -1 -1
## 16     16        16.4.4      4  1  1  1  1
## class=design, type= FrF2.blocked
## NOTE: columns run.no and run.no.std.rp are annotation, not part of the data frame
```

Chapter 4

Two-Level Fractional Factorial Designs

4.1 Generate a 2^{5-2}_{III} design

Original design is a 2^{5-2}III fractional factorial design.

Below, I specify generators I=ABD=ACE. A, B, and C in notes are x_3, x_4, and x_5, respectively. E=AC and below $x_1 = -x_3x_5$, and D=AB and below $x_2 = -x_1x_4$.

```
#### Generate design
library(rsm)

# help for cube, see examples
?cube

# create the first block
block1 <- cube( basis = ~ x1 + x2 + x3 + x4 + x5
              , n0 = 0
              , blockgen = ~ c(x1 * x2 * x4, x1 * x3 * x5)
              , randomize = FALSE
              , bid = 1)
block1

##    run.order std.order x1.as.is x2.as.is x3.as.is x4.as.is x5.as.is
## 1          1         1       -1       -1       -1       -1       -1
## 8          2         2        1        1        1       -1       -1
## 11         3         3       -1        1       -1        1       -1
## 14         4         4        1       -1        1        1       -1
## 20         5         5        1        1       -1       -1        1
## 21         6         6       -1       -1        1       -1        1
## 26         7         7        1       -1       -1        1        1
## 31         8         8       -1        1        1        1        1
##
## Data are stored in coded form using these coding formulas ...
## x1 ~ x1.as.is
## x2 ~ x2.as.is
## x3 ~ x3.as.is
## x4 ~ x4.as.is
## x5 ~ x5.as.is
```

To put this in the same order as in the notes, change the order of the basis.

```
# create the first block
block1 <- cube( basis = ~ x4 + x5 + x1 + x2 + x3
              , n0 = 0
              , blockgen = ~ c(x1 * x2 * x4, x1 * x3 * x5)
              , randomize = FALSE
              , bid = 1)
block1

##    run.order std.order x4.as.is x5.as.is x1.as.is x2.as.is x3.as.is
## 1          1         1       -1       -1       -1       -1       -1
## 8          2         2        1        1        1       -1       -1
## 10         3         3        1       -1       -1        1       -1
## 15         4         4       -1        1        1        1       -1
```

```
## 19          5          5       -1        1       -1       -1        1
## 22          6          6        1       -1        1       -1        1
## 28          7          7        1        1       -1        1        1
## 29          8          8       -1       -1        1        1        1
##
## Data are stored in coded form using these coding formulas ...
## x4 ~ x4.as.is
## x5 ~ x5.as.is
## x1 ~ x1.as.is
## x2 ~ x2.as.is
## x3 ~ x3.as.is
```

4.2 Example 4.5, Table 4.15, p. 161

Original design is a 2^{7-4}III fractional factorial design.

```
#### 4.5a
fn.data <- "http://statacumen.com/teach/RSM/data/RSM_EX_04-05a.txt"
df.4.5a <- read.table(fn.data, header=TRUE)
str(df.4.5a)
```

```
## 'data.frame': 8 obs. of  4 variables:
##  $ a   : int  -1 1 -1 1 -1 1 -1 1
##  $ b   : int  -1 -1 1 1 -1 -1 1 1
##  $ c   : int  -1 -1 -1 -1 1 1 1 1
##  $ time: num  85.5 75.1 93.2 145.4 83.7 ...
```

```
df.4.5a
```

```
##    a  b  c  time
## 1 -1 -1 -1  85.5
## 2  1 -1 -1  75.1
## 3 -1  1 -1  93.2
## 4  1  1 -1 145.4
## 5 -1 -1  1  83.7
## 6  1 -1  1  77.6
## 7 -1  1  1  95.0
## 8  1  1  1 141.8
```

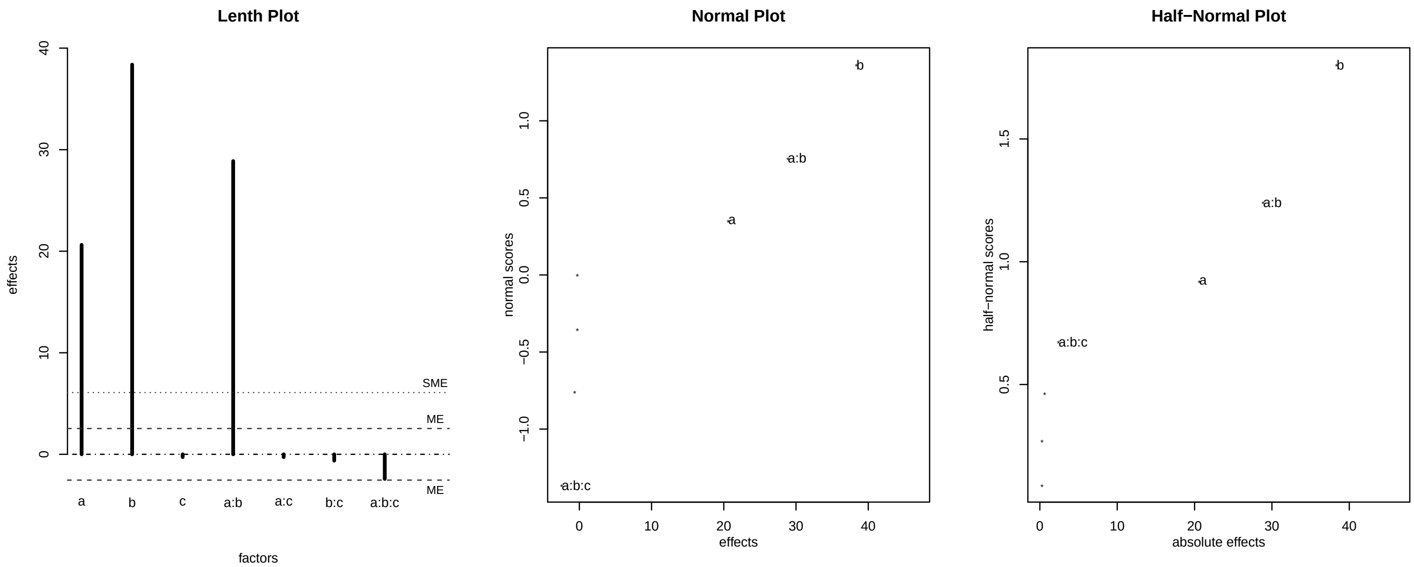
Fit first-order with three-way interaction linear model.

```
lm.4.5a.time.3WIabc <- lm(time ~ (a + b + c)^3, data = df.4.5a)
## externally Studentized residuals
#lm.4.5a.time.3WIabc$studres <- rstudent(lm.4.5a.time.3WIabc)
summary(lm.4.5a.time.3WIabc)
```

```
##
## Call:
## lm.default(formula = time ~ (a + b + c)^3, data = df.4.5a)
##
## Residuals:
## ALL 8 residuals are 0: no residual degrees of freedom!
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   99.662         NA      NA       NA
## a             10.312         NA      NA       NA
## b             19.188         NA      NA       NA
## c             -0.137         NA      NA       NA
## a:b           14.438         NA      NA       NA
## a:c           -0.138         NA      NA       NA
## b:c           -0.313         NA      NA       NA
## a:b:c         -1.212         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
```

The Lenth plot below indicates three large effects worth investigating (a, b, ab).

```r
# BsMD package has unreplicated factorial tests (Daniel plots (aka normal), and Lenth)
library(BsMD)
par(mfrow=c(1,3))
LenthPlot(lm.4.5a.time.3WIabc, alpha = 0.05, main = "Lenth Plot") # , adj = 0.2

## alpha   PSE    ME   SME
## 0.050 0.675 2.541 6.081

DanielPlot(lm.4.5a.time.3WIabc, main = "Normal Plot")
DanielPlot(lm.4.5a.time.3WIabc, half = TRUE, main = "Half-Normal Plot")
```



4.3 Example 4.5, Table 4.16, p. 163

Full fold-over design. Note that the column for a, b, and c are -1 times the same columns in the original design.

```
#### 4.5b
fn.data <- "http://statacumen.com/teach/RSM/data/RSM_EX_04-05b.txt"
df.4.5b <- read.table(fn.data, header=TRUE)
str(df.4.5b)
```

```
## 'data.frame': 8 obs. of  4 variables:
##  $ a   : int  1 -1 1 -1 1 -1 1 -1
##  $ b   : int  1 1 -1 -1 1 1 -1 -1
##  $ c   : int  1 1 1 1 -1 -1 -1 -1
##  $ time: num  91.3 136.7 82.4 73.4 94.1 ...
```

```
df.4.5b
```

```
##    a  b  c  time
## 1  1  1  1  91.3
## 2 -1  1  1 136.7
## 3  1 -1  1  82.4
## 4 -1 -1  1  73.4
## 5  1  1 -1  94.1
## 6 -1  1 -1 143.8
## 7  1 -1 -1  87.3
## 8 -1 -1 -1  71.9
```

Fit first-order with three-way interaction linear model.

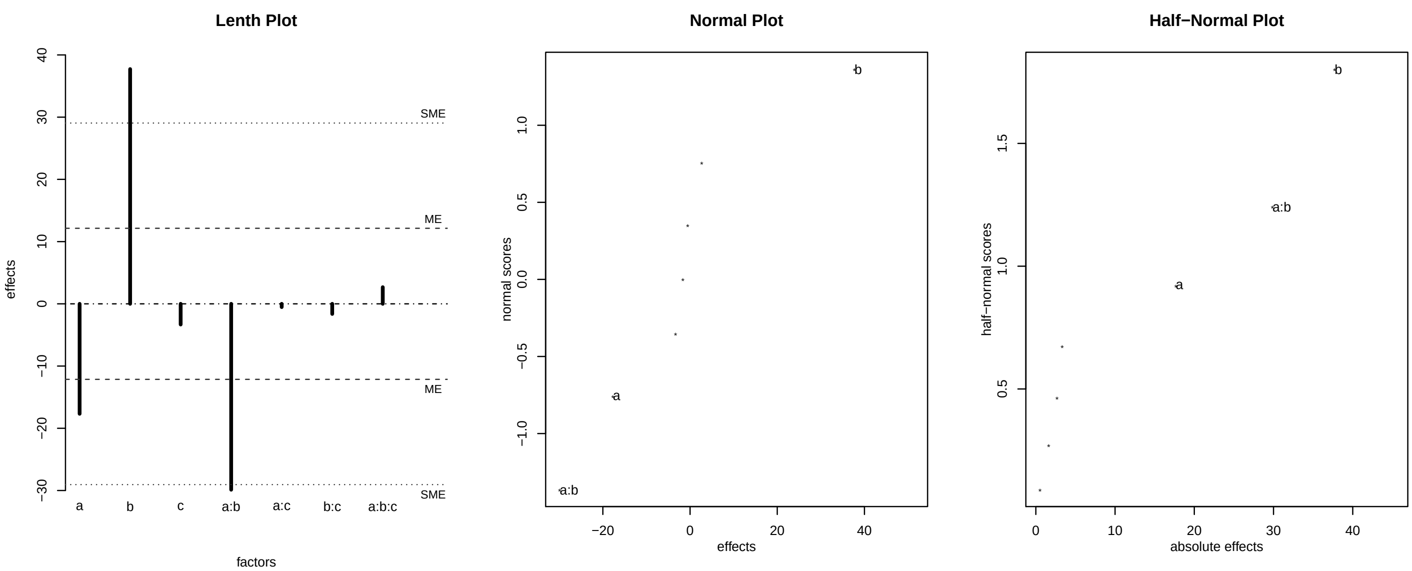
```
lm.4.5b.time.3WIabc <- lm(time ~ (a + b + c)^3, data = df.4.5b)
## externally Studentized residuals
#lm.4.5b.time.3WIabc$studres <- rstudent(lm.4.5b.time.3WIabc)
summary(lm.4.5b.time.3WIabc)
```

```
##
## Call:
## lm.default(formula = time ~ (a + b + c)^3, data = df.4.5b)
##
## Residuals:
## ALL 8 residuals are 0: no residual degrees of freedom!
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   97.612         NA      NA       NA
## a             -8.838         NA      NA       NA
## b             18.862         NA      NA       NA
## c             -1.663         NA      NA       NA
## a:b          -14.938         NA      NA       NA
## a:c           -0.263         NA      NA       NA
## b:c           -0.813         NA      NA       NA
## a:b:c          1.337         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
```

The Lenth plot below indicates the same three large effects are worth investigating (a, b, ab), but some effects are in different directions.

```
# BsMD package has unreplicated factorial tests (Daniel plots (aka normal), and Lenth)
library(BsMD)
par(mfrow=c(1,3))
LenthPlot(lm.4.5b.time.3WIabc, alpha = 0.05, main = "Lenth Plot") # , adj = 0.2

##  alpha    PSE     ME    SME
##  0.050  3.225 12.139 29.052

DanielPlot(lm.4.5b.time.3WIabc, main = "Normal Plot")
DanielPlot(lm.4.5b.time.3WIabc, half = TRUE, main = "Half-Normal Plot")
```



4.4 Combining the data

Combine the two datasets to see what the full factorial suggests.

```
# combine two data sets
df.4.5 <- rbind(df.4.5a, df.4.5b)
str(df.4.5)
```

```
## 'data.frame': 16 obs. of  4 variables:
##  $ a   : int  -1 1 -1 1 -1 1 -1 1 1 -1 ...
##  $ b   : int  -1 -1 1 1 -1 -1 1 1 1 1 ...
##  $ c   : int  -1 -1 -1 -1 1 1 1 1 1 1 ...
##  $ time: num  85.5 75.1 93.2 145.4 83.7 ...
```

```
df.4.5
```

```
##     a  b  c  time
## 1  -1 -1 -1  85.5
## 2   1 -1 -1  75.1
## 3  -1  1 -1  93.2
## 4   1  1 -1 145.4
## 5  -1 -1  1  83.7
## 6   1 -1  1  77.6
## 7  -1  1  1  95.0
## 8   1  1  1 141.8
## 9   1  1  1  91.3
## 10 -1  1  1 136.7
## 11  1 -1  1  82.4
## 12 -1 -1  1  73.4
## 13  1  1 -1  94.1
## 14 -1  1 -1 143.8
## 15  1 -1 -1  87.3
## 16 -1 -1 -1  71.9
```

Fit first-order with three-way interaction linear model.

```
lm.4.5.time.3WIabc <- lm(time ~ (a + b + c)^3, data = df.4.5)
## externally Studentized residuals
#lm.4.5.time.3WIabc£studres <- rstudent(lm.4.5.time.3WIabc)
summary(lm.4.5.time.3WIabc)
```

```
##
## Call:
## lm.default(formula = time ~ (a + b + c)^3, data = df.4.5)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -25.6  -10.3    0.0   10.3   25.6
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  98.6375     6.2325   15.83  2.5e-07 ***
## a             0.7375     6.2325    0.12    0.909
## b            19.0250     6.2325    3.05    0.016 *
## c            -0.9000     6.2325   -0.14    0.889
## a:b          -0.2500     6.2325   -0.04    0.969
```

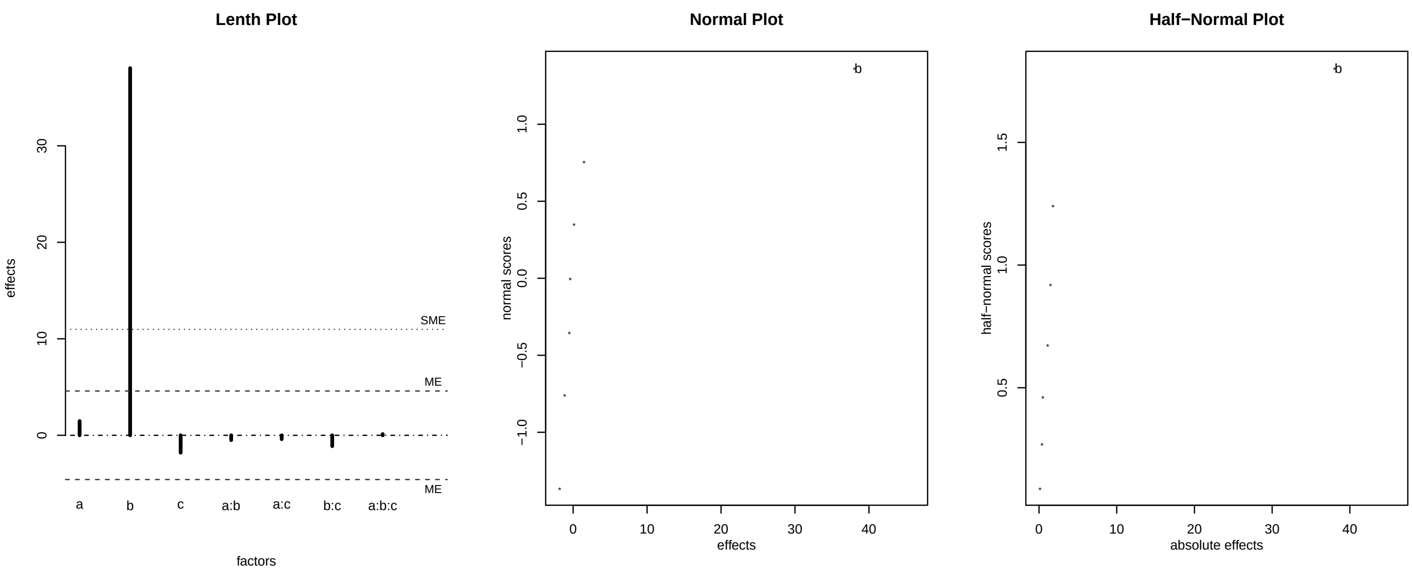
```
## a:c          -0.2000     6.2325   -0.03    0.975
## b:c          -0.5625     6.2325   -0.09    0.930
## a:b:c         0.0625     6.2325    0.01    0.992
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 24.9 on 8 degrees of freedom
## Multiple R-squared:  0.539,Adjusted R-squared:  0.136
## F-statistic: 1.34 on 7 and 8 DF,  p-value: 0.344
```

With more evidence, only effect b is important.

```
# BsMD package has unreplicated factorial tests (Daniel plots (aka normal), and Lenth)
library(BsMD)
par(mfrow=c(1,3))
LenthPlot(lm.4.5.time.3WIabc, alpha = 0.05, main = "Lenth Plot") # , adj = 0.2
```

```
##   alpha    PSE     ME    SME
##   0.050  1.219  4.588 10.979
```

```
DanielPlot(lm.4.5.time.3WIabc, main = "Normal Plot")
DanielPlot(lm.4.5.time.3WIabc, half = TRUE, main = "Half-Normal Plot")
```



4.5 Plackett-Berman design

Here are some examples of Plackett-Berman designs.

```
library(FrF2)
pb(nruns=8, randomize=FALSE)
```

```
## Warning:  Plackett-Burman designs in 8 runs coincide with regular fractional factorials.
##           For screening more than four factors, you may want to consider increasing the
number of runs to 12.
##           Make sure to take the alias structure into account for interpretation!
##    A  B  C  D  E  F  G
## 1  1  1  1 -1  1 -1 -1
## 2 -1  1  1  1 -1  1 -1
## 3 -1 -1  1  1  1 -1  1
## 4  1 -1 -1  1  1  1 -1
## 5 -1  1 -1 -1  1  1  1
## 6  1 -1  1 -1 -1  1  1
## 7  1  1 -1  1 -1 -1  1
## 8 -1 -1 -1 -1 -1 -1 -1
## class=design, type= pb
```

```
pb(nruns=12, randomize=FALSE)
```

```
##     A  B  C  D  E  F  G  H  J  K  L
## 1   1  1 -1  1  1  1 -1 -1 -1  1 -1
## 2  -1  1  1 -1  1  1  1 -1 -1 -1  1
## 3   1 -1  1  1 -1  1  1  1 -1 -1 -1
## 4  -1  1 -1  1  1 -1  1  1  1 -1 -1
## 5  -1 -1  1 -1  1  1 -1  1  1  1 -1
## 6  -1 -1 -1  1 -1  1  1 -1  1  1  1
## 7   1 -1 -1 -1  1 -1  1  1 -1  1  1
## 8   1  1 -1 -1 -1  1 -1  1  1 -1  1
## 9   1  1  1 -1 -1 -1  1 -1  1  1 -1
## 10 -1  1  1  1 -1 -1 -1  1 -1  1  1
## 11  1 -1  1  1  1 -1 -1 -1  1 -1  1
## 12 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
## class=design, type= pb
```

Chapter 5

Process Improvement with Steepest Ascent

5.1 Example 5.1, Table 5.1, p. 185

Build a first-order response function. Read data.

```
#### 5.1
fn.data <- "http://statacumen.com/teach/RSM/data/RSM_EX_05-01.txt"
df.5.1 <- read.table(fn.data, header=TRUE)
str(df.5.1)

## 'data.frame': 8 obs. of  3 variables:
##  $ a: int  -1 1 -1 1 0 0 0 0
##  $ b: int  -1 -1 1 1 0 0 0 0
##  $ y: int  775 670 890 730 745 760 780 720
```

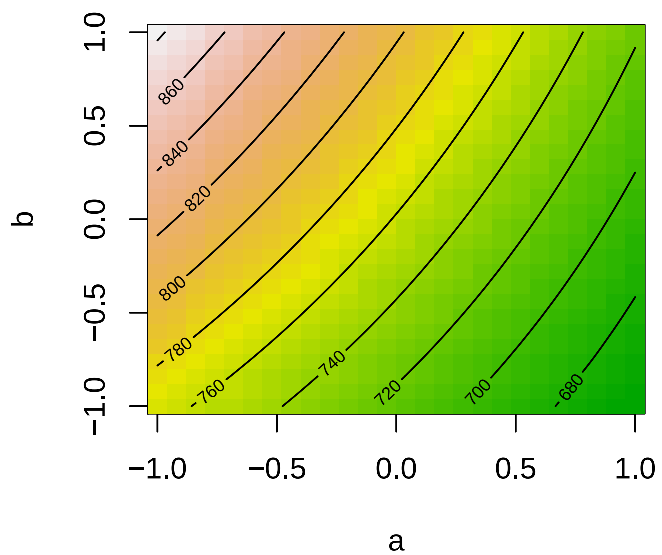
Fit first-order with two-way interaction linear model. This model fit is slightly different than the one in the text; the intercept differs and the significance of the factors.

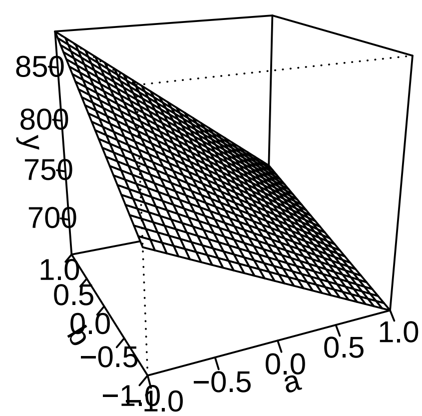
```
library(rsm)
rsm.5.1.y.TWIab <- rsm(y ~ FO(a, b) + TWI(a, b), data = df.5.1)
# externally Studentized residuals
#rsm.5.1.y.TWIab$studres <- rstudent(rsm.5.1.y.TWIab)
summary(rsm.5.1.y.TWIab)

##
## Call:
## rsm(formula = y ~ FO(a, b) + TWI(a, b), data = df.5.1)
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    758.8        8.6   88.19  9.9e-08 ***
## a              -66.2       12.2   -5.44   0.0055 **
## b               43.8       12.2    3.60   0.0228 *
## a:b            -13.8       12.2   -1.13   0.3216
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.916,Adjusted R-squared:  0.854
## F-statistic: 14.6 on 3 and 4 DF,  p-value: 0.0127
##
## Analysis of Variance Table
##
## Response: y
##             Df Sum Sq Mean Sq F value Pr(>F)
## FO(a, b)     2  25212   12606   21.29 0.0074
## TWI(a, b)    1    756     756    1.28 0.3216
## Residuals    4   2369     592
## Lack of fit  1    450     450    0.70 0.4632
## Pure error   3   1919     640
##
## Stationary point of response surface:
##      a      b
##  3.182 -4.818
##
```

```
## Eigenanalysis:
## $values
## [1]  6.875 -6.875
##
## $vectors
##        [,1]    [,2]
## a -0.7071 -0.7071
## b  0.7071 -0.7071
```

Plots indicate the path of steepest ascent is similar to that in Figure 5.3.

```
par(mfrow=c(1,2))
contour(rsm.5.1.y.TWIab, ~ a + b, image = TRUE)
persp(rsm.5.1.y.TWIab, b ~ a, zlab = "y")
```






```
steepest.5.1 <- steepest(rsm.5.1.y.TWIab, dist = seq(0, 7, by = 1))
## Path of steepest ascent from ridge analysis:
steepest.5.1
##   dist      a     b |   yhat
## 1    0  0.000 0.000 |  758.8
## 2    1 -0.805 0.593 |  844.6
## 3    2 -1.573 1.235 |  943.7
## 4    3 -2.321 1.901 | 1056.4
## 5    4 -3.058 2.579 | 1182.6
## 6    5 -3.787 3.265 | 1322.5
## 7    6 -4.511 3.956 | 1476.1
## 8    7 -5.232 4.650 | 1643.3
```

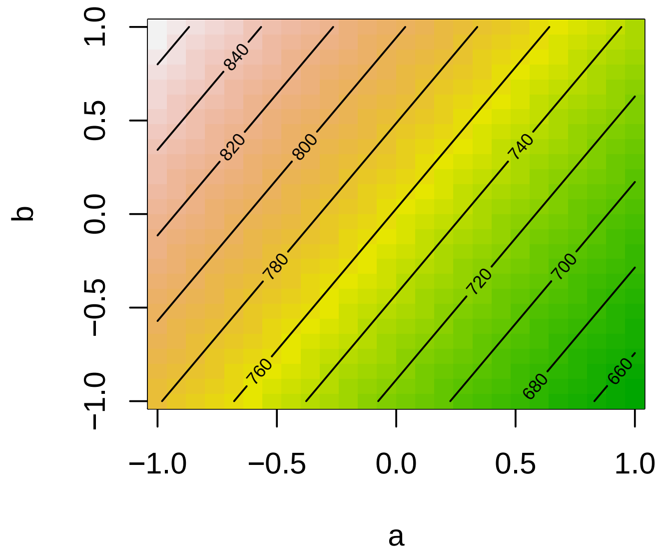
Redo In the text they use a first-order model (equation p. 185) for their steepest ascent calulation.

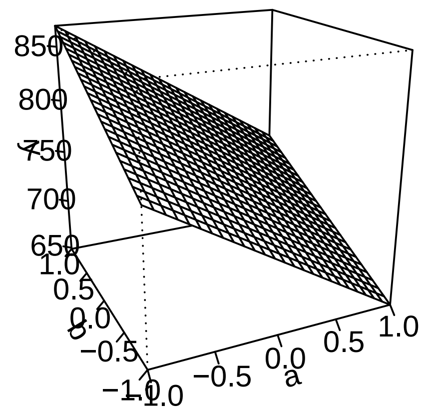
```
library(rsm)
rsm.5.1.y.FOab <- rsm(y ~ FO(a, b), data = df.5.1)
# externally Studentized residuals
#rsm.5.1.y.FOab$studres <- rstudent(rsm.5.1.y.FOab)
summary(rsm.5.1.y.FOab)
```

```
##
## Call:
## rsm(formula = y ~ FO(a, b), data = df.5.1)
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   758.75       8.84    85.8  4.1e-09 ***
## a             -66.25      12.50    -5.3   0.0032 **
## b              43.75      12.50     3.5   0.0173 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.89,Adjusted R-squared:  0.846
## F-statistic: 20.2 on 2 and 5 DF,  p-value: 0.00404
##
## Analysis of Variance Table
##
## Response: y
##             Df Sum Sq Mean Sq F value Pr(>F)
## FO(a, b)     2  25212   12606   20.17  0.004
## Residuals    5   3125     625
## Lack of fit  2   1206     603    0.94  0.481
## Pure error   3   1919     640
##
## Direction of steepest ascent (at radius 1):
##       a       b
## -0.8345  0.5511
##
## Corresponding increment in original units:
##       a       b
## -0.8345  0.5511
```

Plots indicate the path of steepest ascent is similar to that in Figure 5.3.

```
par(mfrow=c(1,2))
contour(rsm.5.1.y.FOab, ~ a + b, image = TRUE)
persp(rsm.5.1.y.FOab, b ~ a, zlab = "y")
```





This result (going in units of a=1) matches Table 5.3.

```
summary(rsm.5.1.y.FOab)$sa

##       a       b
## -0.8345  0.5511

steepest.5.1 <- steepest(rsm.5.1.y.FOab, dist = seq(0, 7, by = 1))

## Path of steepest ascent from ridge analysis:

steepest.5.1$a[2]

## [1] -0.834

steepest.5.1b <- steepest(rsm.5.1.y.FOab, dist = seq(0, 7, by = 1/abs(steepest.5.1$a[2])))

## Path of steepest ascent from ridge analysis:

steepest.5.1b

##    dist      a     b |   yhat
## 1 0.000  0.000 0.000 |  758.8
## 2 1.199 -1.001 0.661 |  854.0
## 3 2.398 -2.001 1.321 |  949.1
## 4 3.597 -3.002 1.982 | 1044.3
## 5 4.796 -4.002 2.643 | 1139.5
## 6 5.995 -5.003 3.304 | 1234.7
```

Chapter 6

The Analysis of Second-Order Response Surfaces

6.1 Example 6.2, Table 6.4, p. 234

Read the data and code the variables. Note that the data are already coded, so I'm including the coding information.

```
#### 6.2
fn.data <- "http://statacumen.com/teach/RSM/data/RSM_EX_06-02.txt"
df.6.2 <- read.table(fn.data, header=TRUE)
df.6.2

##    x1 x2 x3  y
## 1  -1 -1 -1 57
## 2   1 -1 -1 40
## 3  -1  1 -1 19
## 4   1  1 -1 40
## 5  -1 -1  1 54
## 6   1 -1  1 41
## 7  -1  1  1 21
## 8   1  1  1 43
## 9   0  0  0 63
## 10 -2  0  0 28
## 11  2  0  0 11
## 12  0 -2  0  2
## 13  0  2  0 18
## 14  0  0 -2 56
## 15  0  0  2 46

# code the variables
library(rsm)
cd.6.2 <- as.coded.data(df.6.2, x1 ~ (SodiumCitrate     -  3) / 0.7
                              , x2 ~ (Glycerol          -  8) / 3
                              , x3 ~ (EquilibrationTime - 16) / 6)
# the original variables are shown, but their codings are shown
str(cd.6.2)

## Classes 'coded.data' and 'data.frame': 15 obs. of  4 variables:
##  $ x1: int  -1 1 -1 1 -1 1 -1 1 0 -2 ...
##  $ x2: int  -1 -1 1 1 -1 -1 1 1 0 0 ...
##  $ x3: int  -1 -1 -1 -1 1 1 1 1 0 0 ...
##  $ y : int  57 40 19 40 54 41 21 43 63 28 ...
##  - attr(*, "codings")=List of 3
##   ..$ x1:Class 'formula' length 3 x1 ~ (SodiumCitrate - 3)/0.7
##   .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   ..$ x2:Class 'formula' length 3 x2 ~ (Glycerol - 8)/3
##   .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   ..$ x3:Class 'formula' length 3 x3 ~ (EquilibrationTime - 16)/6
##   .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##  - attr(*, "rsdes")=List of 2
##   ..$ primary: chr  "x1" "x2" "x3"
##   ..$ call   : language as.coded.data(data = df.6.2, x1 ~ (SodiumCitrate - 3)/0.7, x2 ~

cd.6.2

##    SodiumCitrate Glycerol EquilibrationTime  y
## 1            2.3        5                10 57
```

```
## 2           3.7        5                 10 40
## 3           2.3       11                 10 19
## 4           3.7       11                 10 40
## 5           2.3        5                 22 54
## 6           3.7        5                 22 41
## 7           2.3       11                 22 21
## 8           3.7       11                 22 43
## 9           3.0        8                 16 63
## 10          1.6        8                 16 28
## 11          4.4        8                 16 11
## 12          3.0        2                 16  2
## 13          3.0       14                 16 18
## 14          3.0        8                  4 56
## 15          3.0        8                 28 46
##
## Data are stored in coded form using these coding formulas ...
## x1 ~ (SodiumCitrate - 3)/0.7
## x2 ~ (Glycerol - 8)/3
## x3 ~ (EquilibrationTime - 16)/6
```

```
# this prints the coded values
print(cd.6.2, decode = FALSE)
```

```
##    x1 x2 x3  y
## 1  -1 -1 -1 57
## 2   1 -1 -1 40
## 3  -1  1 -1 19
## 4   1  1 -1 40
## 5  -1 -1  1 54
## 6   1 -1  1 41
## 7  -1  1  1 21
## 8   1  1  1 43
## 9   0  0  0 63
## 10 -2  0  0 28
## 11  2  0  0 11
## 12  0 -2  0  2
## 13  0  2  0 18
## 14  0  0 -2 56
## 15  0  0  2 46
##
## Variable codings ...
## x1 ~ (SodiumCitrate - 3)/0.7
## x2 ~ (Glycerol - 8)/3
## x3 ~ (EquilibrationTime - 16)/6
```

```
# note that the coded values (-1, +1) are used in the modelling.
```

Fit second-order linear model.

```
library(rsm)
rsm.6.2.y.SOx1x2x3 <- rsm(y ~ SO(x1, x2, x3), data = cd.6.2)
# externally Studentized residuals
rsm.6.2.y.SOx1x2x3$studres <- rstudent(rsm.6.2.y.SOx1x2x3)
summary(rsm.6.2.y.SOx1x2x3)
```

```
##
## Call:
## rsm(formula = y ~ SO(x1, x2, x3), data = cd.6.2)
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   66.111     11.522    5.74   0.0023 **
## x1            -1.312      3.266   -0.40   0.7044
## x2            -2.312      3.266   -0.71   0.5106
## x3            -1.062      3.266   -0.33   0.7581
## x1:x2          9.125      4.619    1.98   0.1052
## x1:x3          0.625      4.619    0.14   0.8976
## x2:x3          0.875      4.619    0.19   0.8572
## x1^2         -11.264      3.925   -2.87   0.0350 *
## x2^2         -13.639      3.925   -3.47   0.0178 *
## x3^2          -3.389      3.925   -0.86   0.4274
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.822,Adjusted R-squared:   0.5
## F-statistic: 2.56 on 9 and 5 DF,  p-value: 0.157
##
## Analysis of Variance Table
##
## Response: y
##                 Df Sum Sq Mean Sq F value Pr(>F)
## FO(x1, x2, x3)   3    131      44    0.26   0.85
## TWI(x1, x2, x3)  3    675     225    1.32   0.37
## PQ(x1, x2, x3)   3   3123    1041    6.10   0.04
## Residuals        5    853     171
## Lack of fit      5    853     171
## Pure error       0      0
##
## Stationary point of response surface:
##      x1      x2      x3
## -0.1158 -0.1294 -0.1841
##
## Stationary point in original units:
##     SodiumCitrate          Glycerol EquilibrationTime
##             2.919             7.612            14.895
##
## Eigenanalysis:
## $values
## [1]  -3.327  -7.797 -17.168
##
## $vectors
##       [,1]    [,2]     [,3]
## x1 0.08493  0.7870  0.61108
## x2 0.07972  0.6060 -0.79149
## x3 0.99319 -0.1159  0.01127
```

```
summary(rsm.6.2.y.SOx1x2x3)$canonical$xs
```

```
##      x1      x2      x3
```

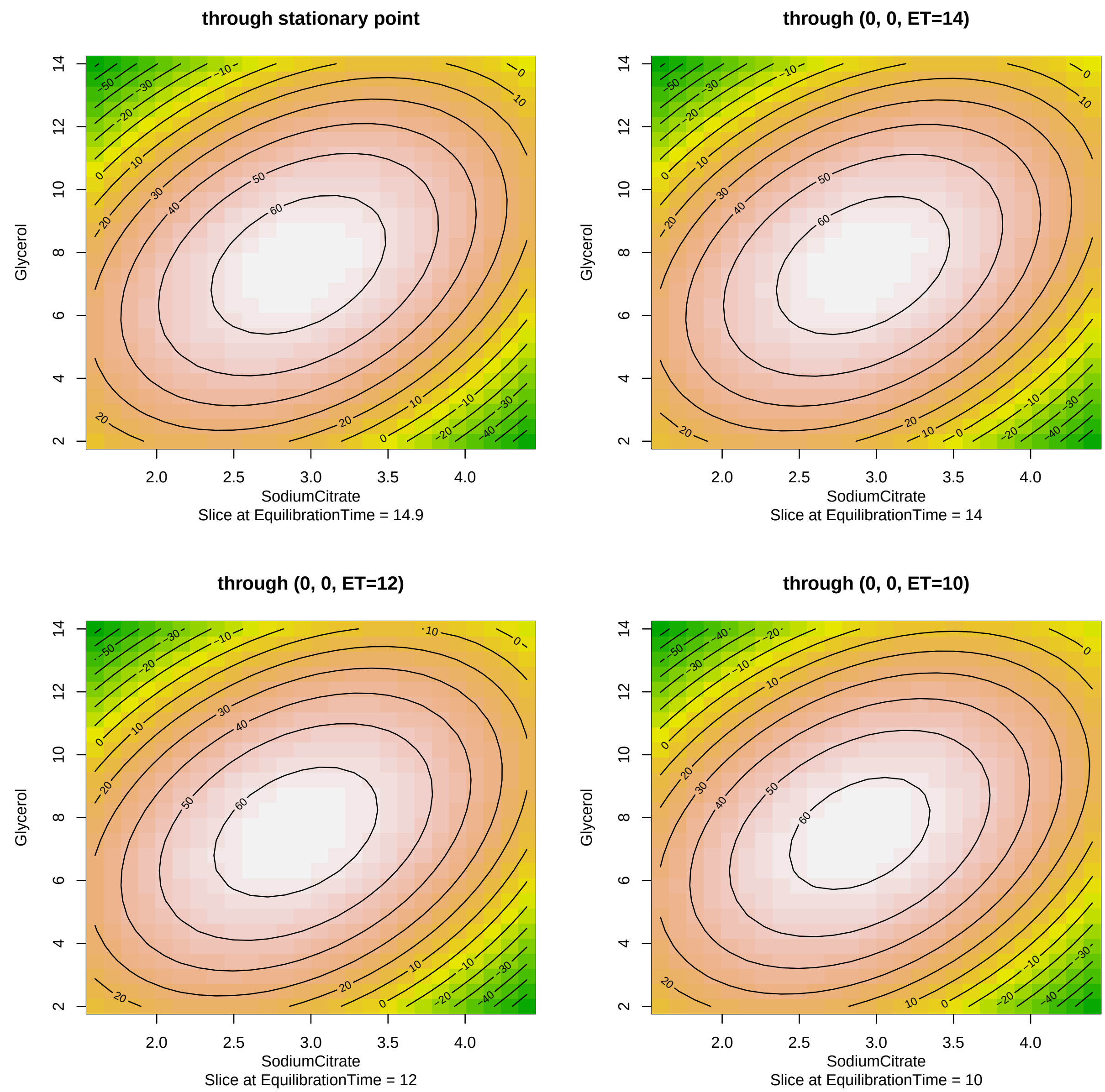
```
## -0.1158 -0.1294 -0.1841
```

The plot below is like that in Figure 6.11. The contour plots below indicates increasing a increases thickness a great deal, c has almost no effect on thickness, and decreasing b increases thickness very little.

```
par(mfrow = c(2,2))
# this is the stationary point
canonical(rsm.6.2.y.SOx1x2x3)$xs
```

```
##      x1      x2      x3
## -0.1158 -0.1294 -0.1841
```

```
contour(rsm.6.2.y.SOx1x2x3, ~ x1 + x2, image=TRUE
      , at = canonical(rsm.6.2.y.SOx1x2x3)$xs, main = "through stationary point")
contour(rsm.6.2.y.SOx1x2x3, ~ x1 + x2, image=TRUE
      , at = data.frame(x1 = 0, x2 = 0, x3 = -1/3), main = "through (0, 0, ET=14)")
contour(rsm.6.2.y.SOx1x2x3, ~ x1 + x2, image=TRUE
      , at = data.frame(x1 = 0, x2 = 0, x3 = -2/3), main = "through (0, 0, ET=12)")
contour(rsm.6.2.y.SOx1x2x3, ~ x1 + x2, image=TRUE
      , at = data.frame(x1 = 0, x2 = 0, x3 = -1  ), main = "through (0, 0, ET=10)")

### Some additional contour plots
# op <- par(no.readonly = TRUE)
# par(mfrow = c(4,3), oma = c(0,0,0,0), mar = c(4,4,2,0))
# contour(rsm.6.2.y.SOx1x2x3, ~ x1 + x2 + x3, image=TRUE, at = data.frame(x1 = 0, x2 = 0, x3
# contour(rsm.6.2.y.SOx1x2x3, ~ x1 + x2 + x3, image=TRUE, at = data.frame(x1 = 0, x2 = 0, x3
# contour(rsm.6.2.y.SOx1x2x3, ~ x1 + x2 + x3, image=TRUE, at = data.frame(x1 = 0, x2 = 0, x3
# contour(rsm.6.2.y.SOx1x2x3, ~ x1 + x2 + x3, image=TRUE, at = canonical(rsm.6.2.y.SOx1x2x3)
# par(op)
```


through stationary point
Glycerol
SodiumCitrate
Slice at EquilibrationTime = 14.9
through (0, 0, ET=14)
Glycerol
SodiumCitrate
Slice at EquilibrationTime = 14
through (0, 0, ET=12)
Glycerol
SodiumCitrate
Slice at EquilibrationTime = 12
through (0, 0, ET=10)
Glycerol
SodiumCitrate
Slice at EquilibrationTime = 10

6.2 Example 6.3, p. 239

Ridge analysis of a saddle point.

Read the data.

```
#### 6.3
fn.data <- "http://statacumen.com/teach/RSM/data/RSM_EX_06-03.txt"
df.6.3 <- read.table(fn.data, header=TRUE)
str(df.6.3)

## 'data.frame': 25 obs. of  5 variables:
##  $ x1: num  -1 1 -1 1 -1 1 -1 1 -1 1 ...
##  $ x2: num  -1 -1 1 1 -1 -1 1 1 -1 -1 ...
##  $ x3: num  -1 -1 -1 -1 1 1 1 1 -1 -1 ...
##  $ x4: num  -1 -1 -1 -1 -1 -1 -1 -1 1 1 ...
##  $ y : num  58.2 23.4 21.9 21.8 14.3 6.3 4.5 21.8 46.7 53.2 ...
```

Fit second-order linear model.

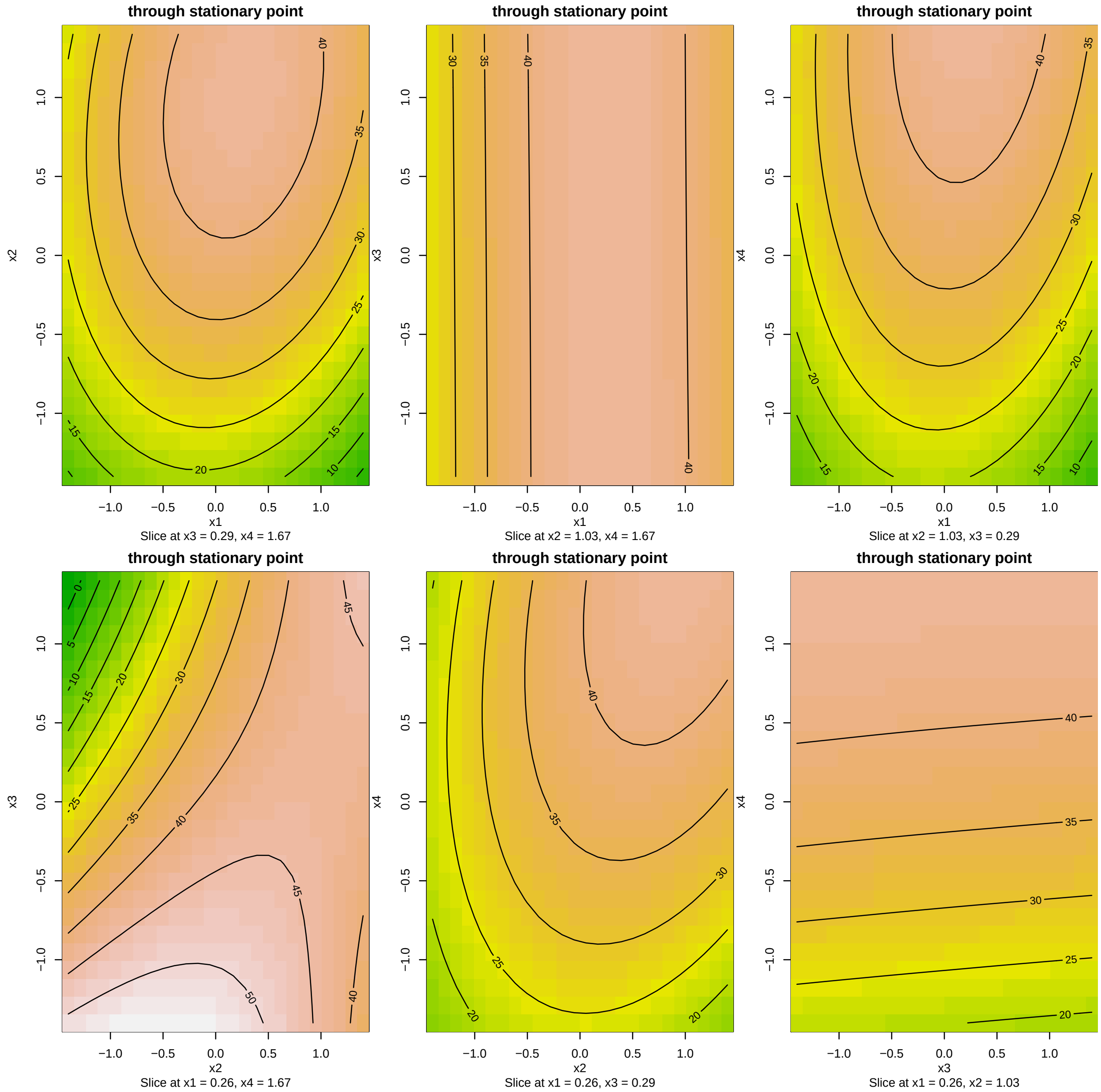
```
library(rsm)
rsm.6.3.y.SOx1x2x3x4 <- rsm(y ~ SO(x1, x2, x3, x4), data = df.6.3)
# externally Studentized residuals
rsm.6.3.y.SOx1x2x3x4$studres <- rstudent(rsm.6.3.y.SOx1x2x3x4)
summary(rsm.6.3.y.SOx1x2x3x4)

##
## Call:
## rsm(formula = y ~ SO(x1, x2, x3, x4), data = df.6.3)
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  40.1982     8.3217    4.83  0.00069 ***
## x1           -1.5110     3.1520   -0.48  0.64197
## x2            1.2841     3.1520    0.41  0.69229
## x3           -8.7390     3.1520   -2.77  0.01970 *
## x4            4.9548     3.1520    1.57  0.14703
## x1:x2         2.1938     3.5170    0.62  0.54675
## x1:x3        -0.1437     3.5170   -0.04  0.96820
## x1:x4         1.5812     3.5170    0.45  0.66258
## x2:x3         8.0062     3.5170    2.28  0.04606 *
## x2:x4         2.8062     3.5170    0.80  0.44345
## x3:x4         0.2937     3.5170    0.08  0.93508
## x1^2         -6.3324     5.0355   -1.26  0.23713
## x2^2         -4.2916     5.0355   -0.85  0.41401
## x3^2          0.0196     5.0355    0.00  0.99696
## x4^2         -2.5059     5.0355   -0.50  0.62950
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.662,Adjusted R-squared:  0.189
## F-statistic:  1.4 on 14 and 10 DF,  p-value: 0.3
##
## Analysis of Variance Table
##
## Response: y
```

```
##                    Df Sum Sq Mean Sq F value Pr(>F)
## FO(x1, x2, x3, x4)   4   2089     522    2.64  0.097
## TWI(x1, x2, x3, x4)  6   1270     212    1.07  0.440
## PQ(x1, x2, x3, x4)   4    520     130    0.66  0.636
## Residuals           10   1979     198
## Lack of fit         10   1979     198
## Pure error           0      0
##
## Stationary point of response surface:
##     x1     x2     x3     x4
## 0.2647 1.0336 0.2906 1.6680
##
## Eigenanalysis:
## $values
## [1]  2.604 -2.159 -6.008 -7.547
##
## $vectors
##         [,1]    [,2]    [,3]     [,4]
## x1 -0.07414  0.2151  0.7688  0.59768
## x2 -0.52824  0.1374  0.4568 -0.70247
## x3 -0.82642 -0.3071 -0.2858  0.37558
## x4 -0.18028  0.9168 -0.3445  0.09085
# the stationary point:
summary(rsm.6.3.y.SOx1x2x3x4)$canonical$xs
##     x1     x2     x3     x4
## 0.2647 1.0336 0.2906 1.6680
canonical(rsm.6.3.y.SOx1x2x3x4)$xs
##     x1     x2     x3     x4
## 0.2647 1.0336 0.2906 1.6680
# just the eigenvalues and eigenvectors
summary(rsm.6.3.y.SOx1x2x3x4)$canonical$eigen
## $values
## [1]  2.604 -2.159 -6.008 -7.547
##
## $vectors
##         [,1]    [,2]    [,3]     [,4]
## x1 -0.07414  0.2151  0.7688  0.59768
## x2 -0.52824  0.1374  0.4568 -0.70247
## x3 -0.82642 -0.3071 -0.2858  0.37558
## x4 -0.18028  0.9168 -0.3445  0.09085
canonical(rsm.6.3.y.SOx1x2x3x4)$eigen
## $values
## [1]  2.604 -2.159 -6.008 -7.547
##
## $vectors
##         [,1]    [,2]    [,3]     [,4]
## x1 -0.07414  0.2151  0.7688  0.59768
## x2 -0.52824  0.1374  0.4568 -0.70247
## x3 -0.82642 -0.3071 -0.2858  0.37558
## x4 -0.18028  0.9168 -0.3445  0.09085
```

The stationary point is just outside region of experimentation for x_4.

The contour plots below go through the stationary point. The saddle is most apparent in the (x_2, x_3) plot.

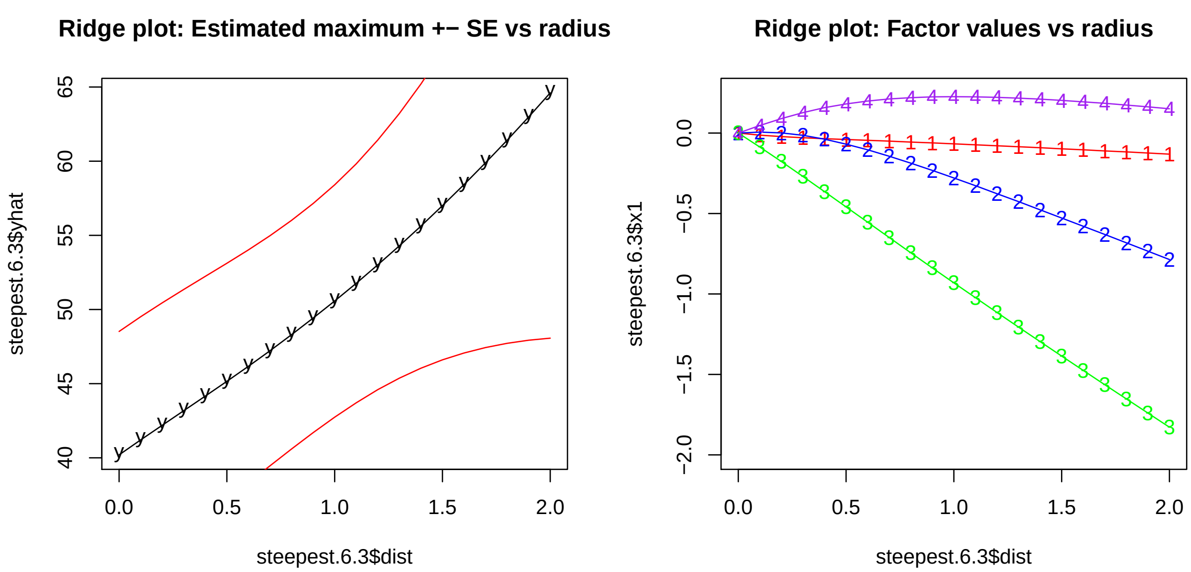
```
par(mfrow = c(2,3), oma = c(0,0,0,0), mar = c(4,4,2,0))
contour(rsm.6.3.y.SOx1x2x3x4, ~ x1 + x2 + x3 + x4, image=TRUE, at = canonical(rsm.6.3.y.SOx1x2
```



Starting at the stationary point, calculate the predicted increase along the ridge of steepest ascent. Include the standard error of the prediction, as well.

```
# calculate predicted increase along ridge of steepest ascent
steepest.6.3 <- steepest(rsm.6.3.y.SOx1x2x3x4, dist = seq(0, 2, by = 0.1))
## Path of steepest ascent from ridge analysis:
# include SE of response in the table, too
predict.6.3 <- predict(rsm.6.3.y.SOx1x2x3x4
                      , newdata = steepest.6.3[,c("x1","x2","x3","x4")], se.fit = TRUE)
steepest.6.3$StdError <- predict.6.3$se.fit
# steepest.6.3
```

```
par(mfrow = c(1, 2))
# plot expected response vs radius
plot  (steepest.6.3$dist, steepest.6.3$yhat, pch = "y"
      , main = "Ridge plot: Estimated maximum +- SE vs radius")
points(steepest.6.3$dist, steepest.6.3$yhat, type = "l")
points(steepest.6.3$dist, steepest.6.3$yhat - predict.6.3$se.fit, type = "l", col = "red")
points(steepest.6.3$dist, steepest.6.3$yhat + predict.6.3$se.fit, type = "l", col = "red")
# plot change of factor variables vs radius
plot  (steepest.6.3$dist, steepest.6.3$x1, pch = "1", col = "red"
      , main = "Ridge plot: Factor values vs radius"
      , ylim = c(-2, 0.25))
points(steepest.6.3$dist, steepest.6.3$x1, type = "l", col = "red")
points(steepest.6.3$dist, steepest.6.3$x2, pch = "2", col = "blue")
points(steepest.6.3$dist, steepest.6.3$x2, type = "l", col = "blue")
points(steepest.6.3$dist, steepest.6.3$x3, pch = "3", col = "green")
points(steepest.6.3$dist, steepest.6.3$x3, type = "l", col = "green")
points(steepest.6.3$dist, steepest.6.3$x4, pch = "4", col = "purple")
points(steepest.6.3$dist, steepest.6.3$x4, type = "l", col = "purple")
```



	dist	x1	x2	x3	x4	yhat	StdError
1	0.000	0.000	0.000	0.000	0.000	40.198	8.322
2	0.100	−0.013	0.006	−0.087	0.047	41.206	8.305
3	0.200	−0.022	0.001	−0.177	0.090	42.193	8.255
4	0.300	−0.029	−0.014	−0.270	0.127	43.177	8.175
5	0.400	−0.035	−0.038	−0.364	0.158	44.162	8.074
6	0.500	−0.040	−0.069	−0.459	0.181	45.156	7.960
7	0.600	−0.045	−0.104	−0.554	0.199	46.171	7.849
8	0.700	−0.050	−0.144	−0.649	0.212	47.218	7.758
9	0.800	−0.056	−0.187	−0.744	0.220	48.302	7.708
10	0.900	−0.061	−0.232	−0.838	0.225	49.420	7.725
11	1.000	−0.067	−0.279	−0.931	0.226	50.572	7.833
12	1.100	−0.073	−0.327	−1.023	0.225	51.764	8.055
13	1.200	−0.079	−0.377	−1.115	0.222	53.012	8.415
14	1.300	−0.085	−0.426	−1.206	0.217	54.292	8.922
15	1.400	−0.091	−0.477	−1.296	0.211	55.621	9.580
16	1.500	−0.098	−0.528	−1.386	0.203	57.002	10.396
17	1.600	−0.104	−0.579	−1.475	0.194	58.421	11.355
18	1.700	−0.111	−0.630	−1.564	0.185	59.895	12.460
19	1.800	−0.117	−0.682	−1.652	0.174	61.411	13.692
20	1.900	−0.124	−0.734	−1.740	0.163	62.983	15.054
21	2.000	−0.131	−0.786	−1.828	0.151	64.608	16.541

6.3 Example from Sec 6.6, Table 6.8, p. 253

Define two functions to obtain maximum or target desirability functions.

```
## Functions for desirability

## Maximum
f.d.max <- function(y, L, T, r) {
  # Computes desirability function (Derringer and Suich)
  #   when object is to maximize the response
  #  y = response
  #  L = unacceptability boundary
  #  T = target acceptability boundary T
  #  r = exponent
  #  d = desirability function

  y.L <- min(T, max(L, y))  # y if L < y, otherwise L, and y if y < T, otherwise T
  d <- ((y.L - L) / (T - L))^r  # desirability function

  return(d)
}

## Target
f.d.target <- function(y, L, T, U, r1, r2) {
  # Computes desirability function (Derringer and Suich)
  # when object is to hit target value
  #  y = response
  #  L = unacceptability boundary
  #  T = target acceptability boundary T
  #  U = upper unacceptability boundary
  # r1 = exponent 1 for L
  # r2 = exponent 2 for U
  #  d = desirability function

  y.L <- min(T, max(L, y))    # y if L < y, otherwise L, and y if y < T, otherwise T
  y.U <- max(T, min(U, y))    # y if y < U, otherwise U, and y if T < y, otherwise T
  d <- (((y.L - L) / (T - L))^r1) * (((U - y.U) / (U - T))^r2)   # desirability function

  return(d)
}
```

Read the data.

```
#### 6.6
fn.data <- "http://statacumen.com/teach/RSM/data/RSM_EX_06-06.txt"
df.6.6 <- read.table(fn.data, header=TRUE)
str(df.6.6)
## 'data.frame': 13 obs. of  5 variables:
##  $ x1: num  -1 1 -1 1 0 ...
##  $ x2: num  -1 -1 1 1 0 0 0 0 0 0 ...
##  $ y1: num  76.5 77 78 79.5 79.9 80.3 80 79.7 79.8 78.4 ...
##  $ y2: int  62 60 66 59 72 69 68 70 71 68 ...
##  $ y3: int  2940 3470 3680 3890 3480 3200 3410 3290 3500 3360 ...
```

Fit second-order linear models for each response variable.

Based on the canonical analysis y_1 has a maximum, y_2 has a maximum, and y_3 has a saddle point.

```
# 6.6, y1
library(rsm)
rsm.6.6.y1.SOx1x2 <- rsm(y1 ~ SO(x1, x2), data = df.6.6)
## externally Studentized residuals
#rsm.6.6.y1.SOx1x2£studres <- rstudent(rsm.6.6.y1.SOx1x2)
#summary(rsm.6.6.y1.SOx1x2)

# 6.6, y2
library(rsm)
rsm.6.6.y2.SOx1x2 <- rsm(y2 ~ SO(x1, x2), data = df.6.6)
## externally Studentized residuals
#rsm.6.6.y2.SOx1x2£studres <- rstudent(rsm.6.6.y2.SOx1x2)
#summary(rsm.6.6.y2.SOx1x2)

# 6.6, y3
library(rsm)
rsm.6.6.y3.SOx1x2 <- rsm(y3 ~ SO(x1, x2), data = df.6.6)
## externally Studentized residuals
#rsm.6.6.y3.SOx1x2£studres <- rstudent(rsm.6.6.y3.SOx1x2)
#summary(rsm.6.6.y3.SOx1x2)

canonical(rsm.6.6.y1.SOx1x2)$eigen$values

## [1] -0.9635 -1.4143

canonical(rsm.6.6.y2.SOx1x2)$eigen$values

## [1] -0.6229 -6.7535

canonical(rsm.6.6.y3.SOx1x2)$eigen$values

## [1]  72.31 -55.77
```

Optimize the response subject to $y_1 \geq 78.5$, $62 \leq y_2 \leq 68$ and $y_3 \leq 3400$. In particular (from p. 259):

```
yield: max      y1: L =   78.5, (T =   85),           r = 1
time:  target   y2: L =   62   ,  T =   65  , U = 68   , r1 = r2 = 1
temp:  min      y3:              (T = 3300), U = 3400, r = 1
```

6.3.1 Method A

Perform RSA on desirability function D evaluated at the **observed** responses at the design points.

For the values in our experiment, calculate D.

```
# Create empty columns to populate with the desirability values
df.6.6$d1 <- NA   #£
df.6.6$d2 <- NA   #£
df.6.6$d3 <- NA   #£
```

```
df.6.6$D  <- NA   #£

# For each data value, calculate desirability
for (i.x in 1:dim(df.6.6)[1]) {
  d1 <- f.d.max   (df.6.6$y1[i.x]
                 , L = 78.5, T = 85, r = 1)
  d2 <- f.d.target(df.6.6$y2[i.x]
                 , L = 62, T = 65, U = 68, r1 = 1, r2 = 1)
  d3 <- f.d.max   (-df.6.6$y3[i.x]
                 , L = -3400, T = -3300, r = 1)
  # Combined desirability
  D <- (d1 * d2 * d3)^(1/3)

  df.6.6[i.x, c("d1", "d2", "d3", "D")] <- c(d1, d2, d3, D)
}

df.6.6

##        x1     x2   y1 y2   y3     d1     d2  d3 D
## 1  -1.000 -1.000 76.5 62 2940 0.0000 0.0000 1.0 0
## 2   1.000 -1.000 77.0 60 3470 0.0000 0.0000 0.0 0
## 3  -1.000  1.000 78.0 66 3680 0.0000 0.6667 0.0 0
## 4   1.000  1.000 79.5 59 3890 0.1538 0.0000 0.0 0
## 5   0.000  0.000 79.9 72 3480 0.2154 0.0000 0.0 0
## 6   0.000  0.000 80.3 69 3200 0.2769 0.0000 1.0 0
## 7   0.000  0.000 80.0 68 3410 0.2308 0.0000 0.0 0
## 8   0.000  0.000 79.7 70 3290 0.1846 0.0000 1.0 0
## 9   0.000  0.000 79.8 71 3500 0.2000 0.0000 0.0 0
## 10 -1.414  0.000 78.4 68 3360 0.0000 0.0000 0.4 0
## 11  1.414  0.000 75.6 71 3020 0.0000 0.0000 1.0 0
## 12  0.000 -1.414 78.5 58 3630 0.0000 0.0000 0.0 0
## 13  0.000  1.414 77.0 57 3150 0.0000 0.0000 1.0 0

# max among these points
df.6.6[which(df.6.6$D == max(df.6.6$D)),]

##        x1     x2   y1 y2   y3     d1     d2  d3 D
## 1  -1.000 -1.000 76.5 62 2940 0.0000 0.0000 1.0 0
## 2   1.000 -1.000 77.0 60 3470 0.0000 0.0000 0.0 0
## 3  -1.000  1.000 78.0 66 3680 0.0000 0.6667 0.0 0
## 4   1.000  1.000 79.5 59 3890 0.1538 0.0000 0.0 0
## 5   0.000  0.000 79.9 72 3480 0.2154 0.0000 0.0 0
## 6   0.000  0.000 80.3 69 3200 0.2769 0.0000 1.0 0
## 7   0.000  0.000 80.0 68 3410 0.2308 0.0000 0.0 0
## 8   0.000  0.000 79.7 70 3290 0.1846 0.0000 1.0 0
## 9   0.000  0.000 79.8 71 3500 0.2000 0.0000 0.0 0
## 10 -1.414  0.000 78.4 68 3360 0.0000 0.0000 0.4 0
## 11  1.414  0.000 75.6 71 3020 0.0000 0.0000 1.0 0
## 12  0.000 -1.414 78.5 58 3630 0.0000 0.0000 0.0 0
## 13  0.000  1.414 77.0 57 3150 0.0000 0.0000 1.0 0
```

However, no overlapping nonzero desirability values, so widening limits on y_1, y_2, and y_3.

```
# Create empty columns to populate with the desirability values
df.6.6$d1 <- NA   #£
df.6.6$d2 <- NA   #£
df.6.6$d3 <- NA   #£
df.6.6$D  <- NA   #£

# For each data value, calculate desirability
for (i.x in 1:dim(df.6.6)[1]) {
  d1 <- f.d.max   (df.6.6$y1[i.x]
                 , L = 70, T = 85, r = 1)
  d2 <- f.d.target(df.6.6$y2[i.x]
                 , L = 58, T = 65, U = 72, r1 = 1, r2 = 1)
  d3 <- f.d.max   (-df.6.6$y3[i.x]
                 , L = -3800, T = -3300, r = 1)
  # Combined desirability
  D <- (d1 * d2 * d3)^(1/3)

  df.6.6[i.x, c("d1", "d2", "d3", "D")] <- c(d1, d2, d3, D)
}

df.6.6

##        x1     x2   y1 y2   y3     d1     d2   d3      D
## 1  -1.000 -1.000 76.5 62 2940 0.4333 0.5714 1.00 0.6280
## 2   1.000 -1.000 77.0 60 3470 0.4667 0.2857 0.66 0.4448
## 3  -1.000  1.000 78.0 66 3680 0.5333 0.8571 0.24 0.4787
## 4   1.000  1.000 79.5 59 3890 0.6333 0.1429 0.00 0.0000
## 5   0.000  0.000 79.9 72 3480 0.6600 0.0000 0.64 0.0000
## 6   0.000  0.000 80.3 69 3200 0.6867 0.4286 1.00 0.6652
## 7   0.000  0.000 80.0 68 3410 0.6667 0.5714 0.78 0.6673
## 8   0.000  0.000 79.7 70 3290 0.6467 0.2857 1.00 0.5696
## 9   0.000  0.000 79.8 71 3500 0.6533 0.1429 0.60 0.3826
## 10 -1.414  0.000 78.4 68 3360 0.5600 0.5714 0.88 0.6555
## 11  1.414  0.000 75.6 71 3020 0.3733 0.1429 1.00 0.3764
## 12  0.000 -1.414 78.5 58 3630 0.5667 0.0000 0.34 0.0000
## 13  0.000  1.414 77.0 57 3150 0.4667 0.0000 1.00 0.0000

# max among these points
df.6.6[which(df.6.6$D == max(df.6.6$D)),]

##   x1 x2 y1 y2   y3     d1     d2   d3      D
## 7  0  0 80 68 3410 0.6667 0.5714 0.78 0.6673
```

Now fit a response surface for D to predict the optimal conditions.

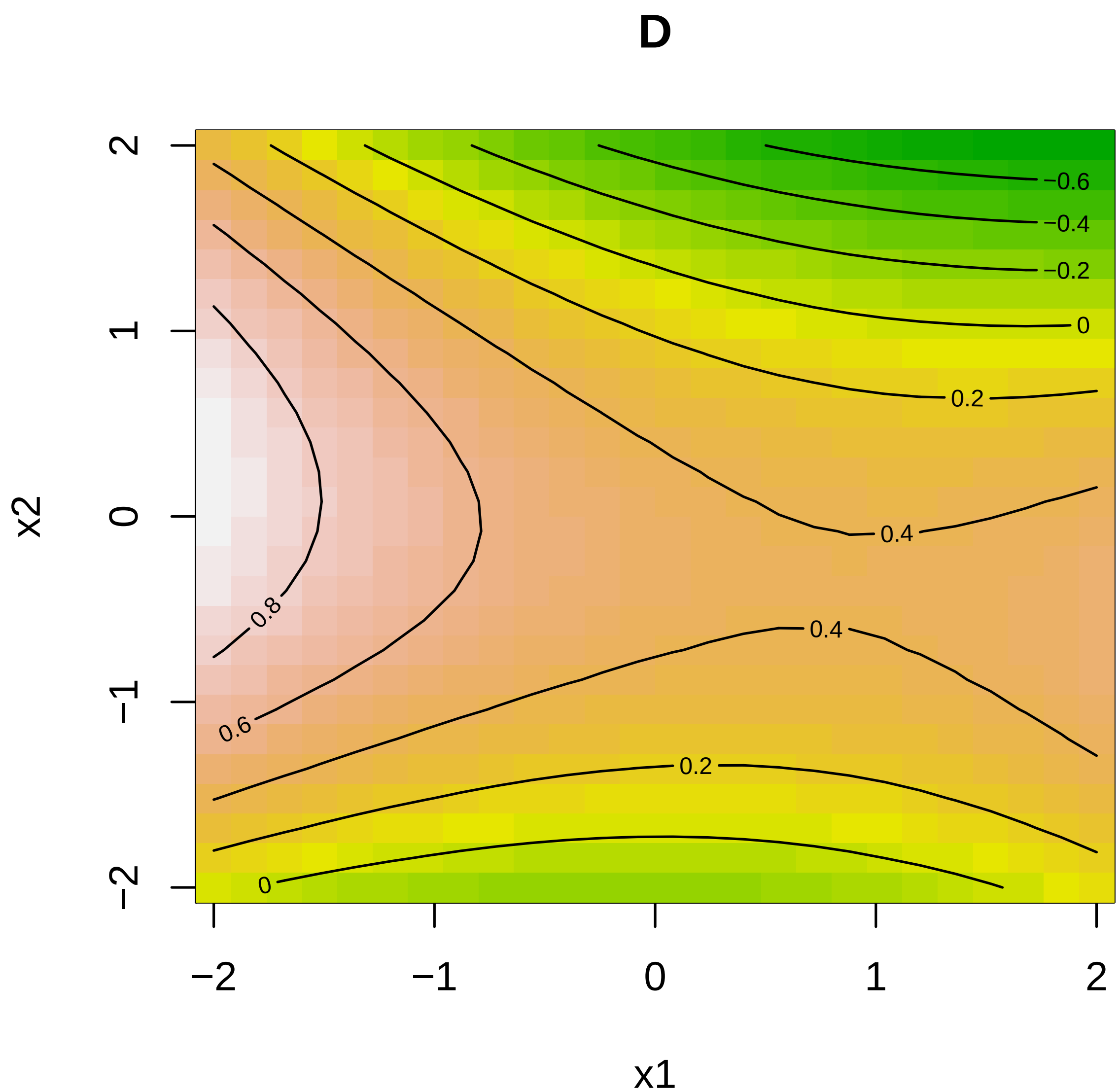
```
# D as response
library(rsm)
rsm.6.6.D.SOx1x2 <- rsm(D ~ SO(x1, x2), data = df.6.6)
# externally Studentized residuals
rsm.6.6.D.SOx1x2$studres <- rstudent(rsm.6.6.D.SOx1x2)
summary(rsm.6.6.D.SOx1x2)

##
## Call:
## rsm(formula = D ~ SO(x1, x2), data = df.6.6)
##
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.4569     0.1071    4.27   0.0037 **
## x1           -0.1321     0.0847   -1.56   0.1628
## x2           -0.0743     0.0847   -0.88   0.4095
## x1:x2        -0.0739     0.1197   -0.62   0.5567
## x1^2          0.0620     0.0908    0.68   0.5166
## x2^2         -0.1960     0.0908   -2.16   0.0678 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.568,Adjusted R-squared:  0.259
## F-statistic: 1.84 on 5 and 7 DF,  p-value: 0.224
##
## Analysis of Variance Table
##
## Response: D
##             Df Sum Sq Mean Sq F value Pr(>F)
## FO(x1, x2)   2  0.184  0.0918    1.60   0.27
## TWI(x1, x2)  1  0.022  0.0218    0.38   0.56
## PQ(x1, x2)   2  0.321  0.1607    2.80   0.13
## Residuals    7  0.401  0.0573
## Lack of fit  3  0.087  0.0289    0.37   0.78
## Pure error   4  0.315  0.0787
##
## Stationary point of response surface:
##      x1      x2
##  0.8558 -0.3507
##
## Eigenanalysis:
## $values
## [1]  0.06721 -0.20121
##
## $vectors
##        [,1]   [,2]
## x1 -0.9903 0.1390
## x2  0.1390 0.9903
```

This model is insignificant for lack-of-fit.

The contour plot for D is below.

```
par(mfrow = c(1,1))
contour(rsm.6.6.D.SOx1x2, ~ x1 + x2
      , bounds = list(x1 = c(-2, 2), x2 = c(-2, 2))
      , image=TRUE, main = "D")
```



Summary: D has all zeros for original values. The RSA gives a saddle point for wider bounds for y_1, y_2, and y_3.

Method A2

A brute-force search for the optimum predicted deirability, D, over the ± 2-unit cube. gives the result below.

```
# x-values
D.cube <- expand.grid(seq(-2, 2, by=0.1), seq(-2, 2, by=0.1))
colnames(D.cube) <- c("x1", "x2")
# predicted D
D.cube$D <- predict(rsm.6.6.D.SOx1x2, newdata = D.cube)

# predicted optimum
D.opt <- D.cube[which(D.cube$D == max(D.cube$D)),]
```

```
D.opt
```

```
##     x1  x2     D
## 903 -2 0.2 0.976
```

```
# predicted response at the optimal D
y1 <- predict(rsm.6.6.y1.SOx1x2, newdata = D.opt)
y2 <- predict(rsm.6.6.y2.SOx1x2, newdata = D.opt)
y3 <- predict(rsm.6.6.y3.SOx1x2, newdata = D.opt)
c(y1, y2, y3)
```

```
##     903     903     903
##   74.83   68.71 3190.55
```

Always check that the optimal value is contained in the specified constraints.

Summary: D has all zeros for original values. The RSA gives a saddle point for wider bounds for y_1, y_2, and y_3.

6.3.2 Method B

Perform RSA on desirability function D evaluated at the **predicted** responses at the design points.

At the center point, these are the desirability values, and the combined desirability, D.

```
d1 <- f.d.max   (predict(rsm.6.6.y1.SOx1x2, newdata = data.frame(x1=0, x2=0, x3=0))
                , L = 78.5, T = 85, r = 1)
d2 <- f.d.target(predict(rsm.6.6.y2.SOx1x2, newdata = data.frame(x1=0, x2=0, x3=0))
                , L = 62, T = 65, U = 68, r1 = 1, r2 = 1)
d3 <- f.d.max   (-predict(rsm.6.6.y3.SOx1x2, newdata = data.frame(x1=0, x2=0, x3=0))
                , L = -3400, T = -3300, r = 1)

# Combined desirability
D <- (d1 * d2 * d3)^(1/3)
c(d1, d2, d3, D)
```

```
## [1] 0.2215 0.0000 0.2402 0.0000
```

For the values in our experiment, calculate D.

```
# Create empty columns to populate with the desirability values
df.6.6$d1 <- NA   #£
df.6.6$d2 <- NA   #£
df.6.6$d3 <- NA   #£
df.6.6$D  <- NA   #£

# For each data value, calculate desirability
for (i.x in 1:dim(df.6.6)[1]) {
  d1 <- f.d.max   (predict(rsm.6.6.y1.SOx1x2, newdata = data.frame(x1=df.6.6$x1[i.x], x2=df.6.
                  , L = 78.5, T = 85, r = 1)
  d2 <- f.d.target(predict(rsm.6.6.y2.SOx1x2, newdata = data.frame(x1=df.6.6$x1[i.x], x2=df.6.
                  , L = 62, T = 65, U = 68, r1 = 1, r2 = 1)
```

```
  d3 <- f.d.max   (-predict(rsm.6.6.y3.SOx1x2, newdata = data.frame(x1=df.6.6$x1[i.x], x2=df.6
                 , L = -3400, T = -3300, r = 1)
  # Combined desirability
  D <- (d1 * d2 * d3)^(1/3)

  df.6.6[i.x, c("d1", "d2", "d3", "D")] <- c(d1, d2, d3, D)
}

df.6.6
```

```
##        x1     x2   y1 y2   y3     d1      d2     d3 D
## 1  -1.000 -1.000 76.5 62 2940 0.0000 0.00000 1.0000 0
## 2   1.000 -1.000 77.0 60 3470 0.0000 0.36021 0.0000 0
## 3  -1.000  1.000 78.0 66 3680 0.0000 0.88896 0.0000 0
## 4   1.000  1.000 79.5 59 3890 0.0000 0.00000 0.0000 0
## 5   0.000  0.000 79.9 72 3480 0.2215 0.00000 0.2402 0
## 6   0.000  0.000 80.3 69 3200 0.2215 0.00000 0.2402 0
## 7   0.000  0.000 80.0 68 3410 0.2215 0.00000 0.2402 0
## 8   0.000  0.000 79.7 70 3290 0.2215 0.00000 0.2402 0
## 9   0.000  0.000 79.8 71 3500 0.2215 0.00000 0.2402 0
## 10 -1.414  0.000 78.4 68 3360 0.0000 0.00000 1.0000 0
## 11  1.414  0.000 75.6 71 3020 0.0000 0.07171 0.6166 0
## 12  0.000 -1.414 78.5 58 3630 0.0000 0.00000 0.0000 0
## 13  0.000  1.414 77.0 57 3150 0.0000 0.00000 0.0000 0
```

```
# max among these points
df.6.6[which(df.6.6$D == max(df.6.6$D)),]
```

```
##        x1     x2   y1 y2   y3     d1      d2     d3 D
## 1  -1.000 -1.000 76.5 62 2940 0.0000 0.00000 1.0000 0
## 2   1.000 -1.000 77.0 60 3470 0.0000 0.36021 0.0000 0
## 3  -1.000  1.000 78.0 66 3680 0.0000 0.88896 0.0000 0
## 4   1.000  1.000 79.5 59 3890 0.0000 0.00000 0.0000 0
## 5   0.000  0.000 79.9 72 3480 0.2215 0.00000 0.2402 0
## 6   0.000  0.000 80.3 69 3200 0.2215 0.00000 0.2402 0
## 7   0.000  0.000 80.0 68 3410 0.2215 0.00000 0.2402 0
## 8   0.000  0.000 79.7 70 3290 0.2215 0.00000 0.2402 0
## 9   0.000  0.000 79.8 71 3500 0.2215 0.00000 0.2402 0
## 10 -1.414  0.000 78.4 68 3360 0.0000 0.00000 1.0000 0
## 11  1.414  0.000 75.6 71 3020 0.0000 0.07171 0.6166 0
## 12  0.000 -1.414 78.5 58 3630 0.0000 0.00000 0.0000 0
## 13  0.000  1.414 77.0 57 3150 0.0000 0.00000 0.0000 0
```

However, no overlapping nonzero desirability values, so widening limits on y_1, y_2, and y_3.

```
# Create empty columns to populate with the desirability values
df.6.6$d1 <- NA   #£
df.6.6$d2 <- NA   #£
df.6.6$d3 <- NA   #£
df.6.6$D  <- NA   #£

# For each data value, calculate desirability
for (i.x in 1:dim(df.6.6)[1]) {
  d1 <- f.d.max   (predict(rsm.6.6.y1.SOx1x2, newdata = data.frame(x1=df.6.6$x1[i.x], x2=df.6.
```

```
                 , L = 70, T = 85, r = 1)
  d2 <- f.d.target(predict(rsm.6.6.y2.SOx1x2, newdata = data.frame(x1=df.6.6$x1[i.x], x2=df.6.
                 , L = 58, T = 65, U = 72, r1 = 1, r2 = 1)
  d3 <- f.d.max   (-predict(rsm.6.6.y3.SOx1x2, newdata = data.frame(x1=df.6.6$x1[i.x], x2=df.6
                 , L = -3800, T = -3300, r = 1)
  # Combined desirability
  D <- (d1 * d2 * d3)^(1/3)

  df.6.6[i.x, c("d1", "d2", "d3", "D")] <- c(d1, d2, d3, D)
}

df.6.6

##        x1     x2   y1 y2   y3     d1     d2     d3      D
## 1  -1.000 -1.000 76.5 62 2940 0.5215 0.5386 1.0000 0.6549
## 2   1.000 -1.000 77.0 60 3470 0.4555 0.7258 0.7105 0.6170
## 3  -1.000  1.000 78.0 66 3680 0.5195 0.9524 0.5994 0.6669
## 4   1.000  1.000 79.5 59 3890 0.5201 0.4253 0.7898 0.5591
## 5   0.000  0.000 79.9 72 3480 0.6627 0.2857 0.8480 0.5435
## 6   0.000  0.000 80.3 69 3200 0.6627 0.2857 0.8480 0.5435
## 7   0.000  0.000 80.0 68 3410 0.6627 0.2857 0.8480 0.5435
## 8   0.000  0.000 79.7 70 3290 0.6627 0.2857 0.8480 0.5435
## 9   0.000  0.000 79.8 71 3500 0.6627 0.2857 0.8480 0.5435
## 10 -1.414  0.000 78.4 68 3360 0.5023 0.3618 1.0000 0.5664
## 11  1.414  0.000 75.6 71 3020 0.4561 0.6022 0.9233 0.6330
## 12  0.000 -1.414 78.5 58 3630 0.5070 0.0000 0.7851 0.0000
## 13  0.000  1.414 77.0 57 3150 0.5513 0.0000 0.4448 0.0000

# max among these points
df.6.6[which(df.6.6$D == max(df.6.6$D)),]

##   x1 x2 y1 y2   y3     d1     d2     d3      D
## 3 -1  1 78 66 3680 0.5195 0.9524 0.5994 0.6669
```

Now fit a response surface for D to predict the optimal conditions.

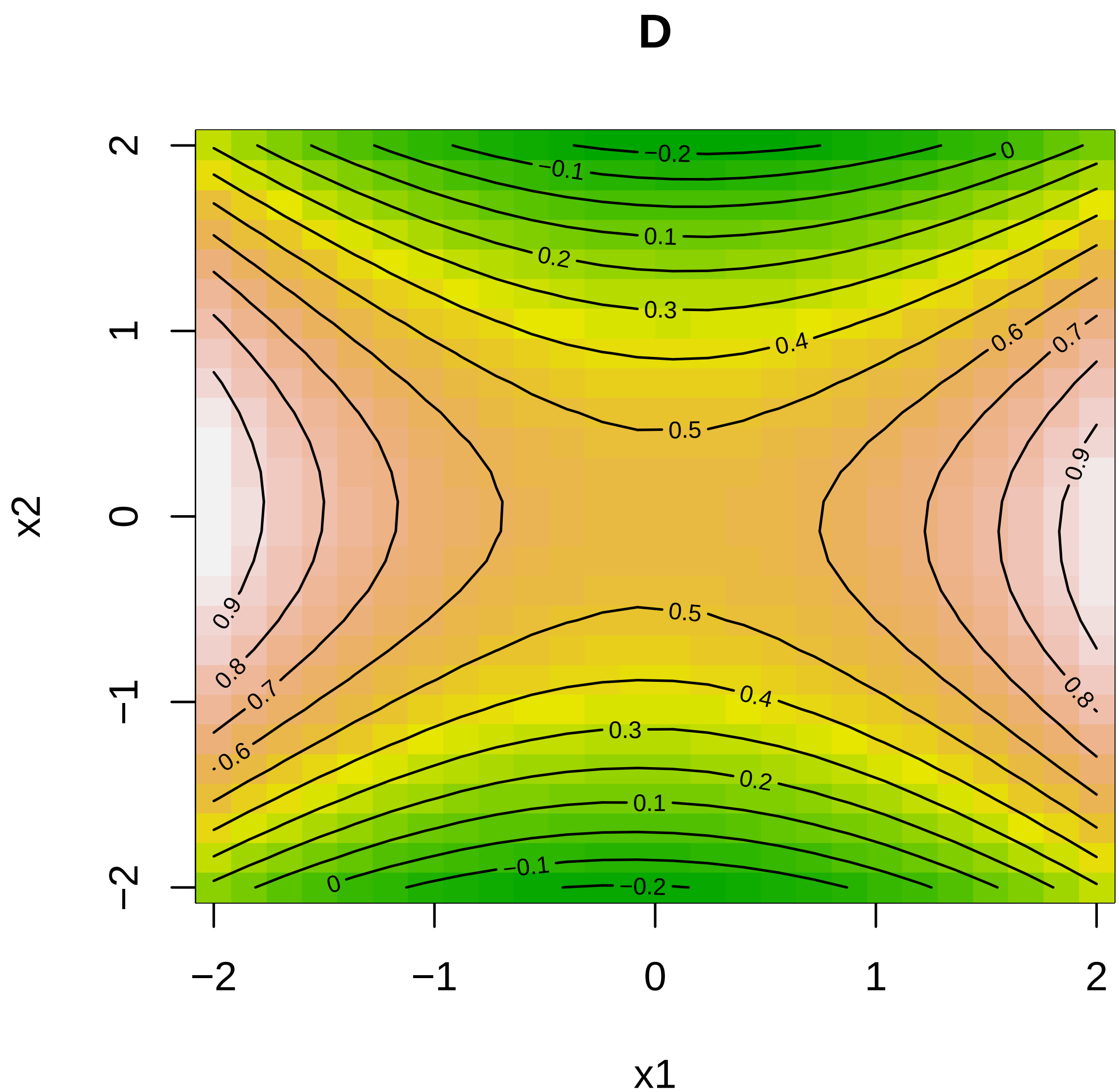
```
# D as response
library(rsm)
rsm.6.6.D.SOx1x2 <- rsm(D ~ SO(x1, x2), data = df.6.6)
# externally Studentized residuals
rsm.6.6.D.SOx1x2$studres <- rstudent(rsm.6.6.D.SOx1x2)
summary(rsm.6.6.D.SOx1x2)

##
## Call:
## rsm(formula = D ~ SO(x1, x2), data = df.6.6)
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.54346    0.07896    6.88  0.00023 ***
## x1          -0.00646    0.06243   -0.10  0.92049
## x2          -0.00575    0.06243   -0.09  0.92923
## x1:x2       -0.01748    0.08828   -0.20  0.84866
## x1^2         0.10932    0.06696    1.63  0.14654
## x2^2        -0.19062    0.06696   -2.85  0.02480 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Multiple R-squared:  0.636,Adjusted R-squared:  0.377
## F-statistic: 2.45 on 5 and 7 DF,  p-value: 0.137
##
## Analysis of Variance Table
##
## Response: D
##             Df Sum Sq Mean Sq  F value Pr(>F)
## FO(x1, x2)   2  0.001  0.0003 1.00e-02  0.990
## TWI(x1, x2)  1  0.001  0.0012 4.00e-02  0.849
## PQ(x1, x2)   2  0.380  0.1900 6.09e+00  0.029
## Residuals    7  0.218  0.0312
## Lack of fit  3  0.218  0.0727 2.06e+31 <2e-16
## Pure error   4  0.000  0.0000
##
## Stationary point of response surface:
##       x1       x2
##  0.02824 -0.01637
##
## Eigenanalysis:
## $values
## [1]  0.1096 -0.1909
##
## $vectors
##       [,1]   [,2]
## x1 -0.9996 0.0291
## x2  0.0291 0.9996
```

This model has severe lack-of-fit, but we'll continue anyway.

The contour plot for D is below.

```
par(mfrow = c(1,1))
contour(rsm.6.6.D.SOx1x2, ~ x1 + x2
      , bounds = list(x1 = c(-2, 2), x2 = c(-2, 2))
      , image=TRUE, main = "D")
```



Method C

A brute-force search for the optimum predicted deirability, D, over the ± 2-unit cube. gives the result below.

```
# x-values
D.cube <- expand.grid(seq(-2, 2, by=0.1), seq(-2, 2, by=0.1))
colnames(D.cube) <- c("x1", "x2")
# predicted D
D.cube$D <- predict(rsm.6.6.D.SOx1x2, newdata = D.cube)

# predicted optimum
D.opt <- D.cube[which(D.cube$D == max(D.cube$D)),]
D.opt
```

```
##      x1  x2      D
## 862 -2 0.1 0.9947
```

```
# predicted response at the optimal D
y1 <- predict(rsm.6.6.y1.SOx1x2, newdata = D.opt)
y2 <- predict(rsm.6.6.y2.SOx1x2, newdata = D.opt)
y3 <- predict(rsm.6.6.y3.SOx1x2, newdata = D.opt)
c(y1, y2, y3)

##     862     862     862
##   74.89   68.64 3166.78
```

Always check that the optimal value is contained in the specified constraints.

Summary: D has all zeros for original values. The RSA gives a saddle point for wider bounds for y_1, y_2, and y_3.

6.4 Example 6.8 from 2nd edition – Box-Cox transformation

Read the data.

```
#### 6.8
fn.data <- "http://statacumen.com/teach/RSM/data/RSM_EX_06-08.txt"
df.6.8 <- read.table(fn.data, header=TRUE)
str(df.6.8)
```

```
## 'data.frame': 27 obs. of  4 variables:
##  $ x1: int  -1 0 1 -1 0 1 -1 0 1 -1 ...
##  $ x2: int  -1 -1 -1 0 0 0 1 1 1 -1 ...
##  $ x3: int  -1 -1 -1 -1 -1 -1 -1 -1 -1 0 ...
##  $ y : int  674 1414 3636 338 1022 1368 170 442 1140 370 ...
```

Fit second-order linear model.

```
library(rsm)
rsm.6.8.y.SOx1x2x3 <- rsm(y ~ SO(x1, x2, x3), data = df.6.8)
# externally Studentized residuals
rsm.6.8.y.SOx1x2x3$studres <- rstudent(rsm.6.8.y.SOx1x2x3)
summary(rsm.6.8.y.SOx1x2x3)
```

```
##
## Call:
## rsm(formula = y ~ SO(x1, x2, x3), data = df.6.8)
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    543.3      147.9    3.67  0.00189 **
## x1             648.9       68.5    9.48  3.4e-08 ***
## x2            -535.9       68.5   -7.83  4.9e-07 ***
## x3            -299.7       68.5   -4.38  0.00041 ***
## x1:x2         -456.5       83.9   -5.44  4.4e-05 ***
## x1:x3         -219.0       83.9   -2.61  0.01825 *
## x2:x3          143.0       83.9    1.71  0.10637
## x1^2           227.6      118.6    1.92  0.07198 .
## x2^2           297.9      118.6    2.51  0.02241 *
## x3^2           -59.4      118.6   -0.50  0.62265
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.928,Adjusted R-squared:  0.89
## F-statistic: 24.4 on 9 and 17 DF,  p-value: 5.17e-08
##
## Analysis of Variance Table
##
## Response: y
##                 Df   Sum Sq Mean Sq F value  Pr(>F)
## FO(x1, x2, x3)   3 14364608 4788203   56.73 4.6e-09
## TWI(x1, x2, x3)  3  3321627 1107209   13.12 0.00011
## PQ(x1, x2, x3)   3   864318  288106    3.41 0.04137
## Residuals       17  1434755   84397
## Lack of fit     17  1434755   84397
```

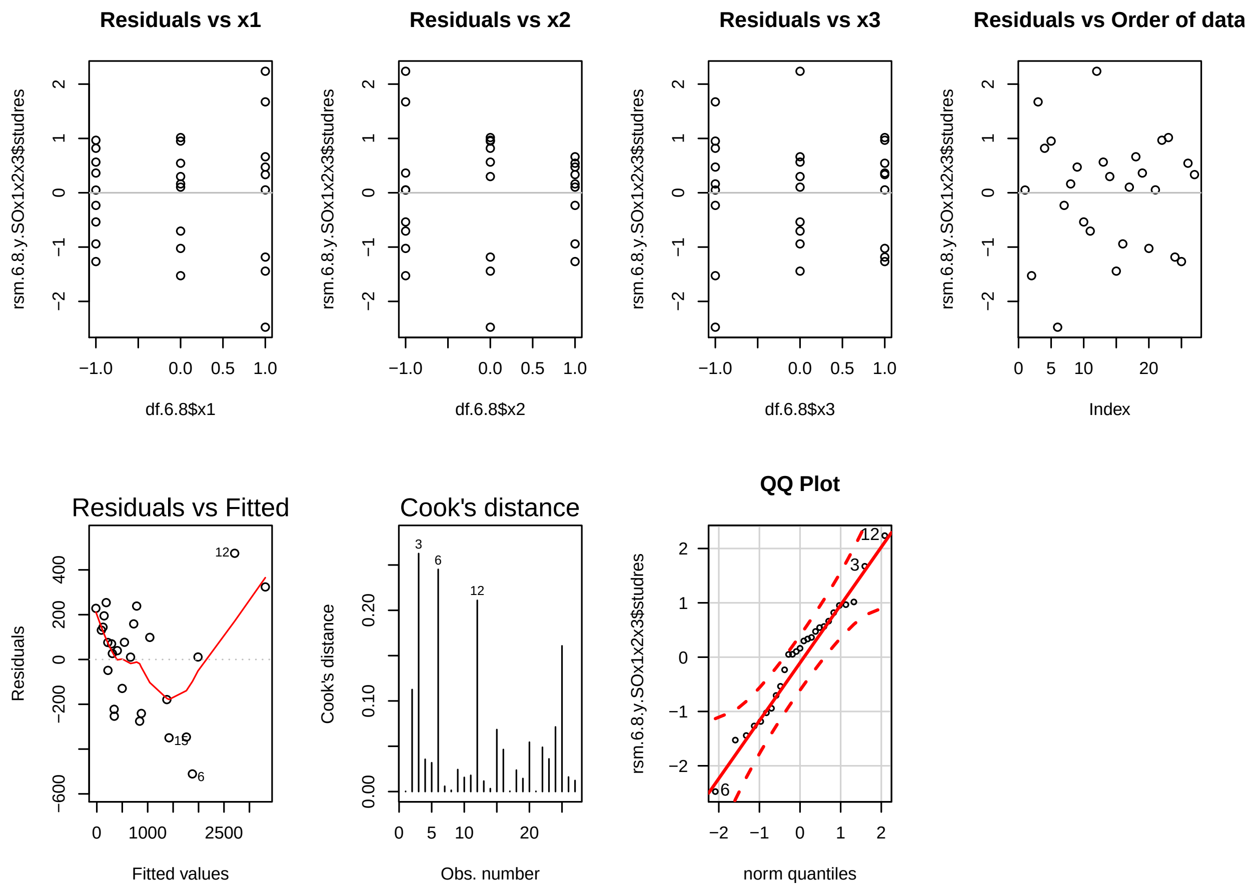
```
## Pure error       0       0
##
## Stationary point of response surface:
##      x1      x2      x3
## -1.9644 -0.6745  0.2867
##
## Eigenanalysis:
## $values
## [1] 520.96  41.61 -96.57
##
## $vectors
##        [,1]    [,2]    [,3]
## x1  0.6482  0.6842 0.33424
## x2 -0.7313  0.6817 0.02261
## x3 -0.2124 -0.2591 0.94222
```

This second-order model fits pretty well. Note there are many interaction and second-order terms.

The residual plots do not indicate a problem with normality of these residuals.

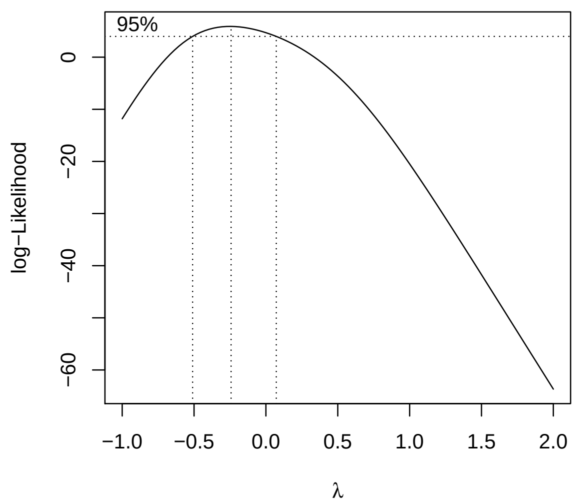
```
# plot diagnostics
par(mfrow=c(2,4))

plot(df.6.8$x1, rsm.6.8.y.SOx1x2x3$studres, main="Residuals vs x1")
  # horizontal line at zero
  abline(h = 0, col = "gray75")

plot(df.6.8$x2, rsm.6.8.y.SOx1x2x3$studres, main="Residuals vs x2")
  # horizontal line at zero
  abline(h = 0, col = "gray75")

plot(df.6.8$x3, rsm.6.8.y.SOx1x2x3$studres, main="Residuals vs x3")
  # horizontal line at zero
  abline(h = 0, col = "gray75")

# residuals vs order of data
plot(rsm.6.8.y.SOx1x2x3$studres, main="Residuals vs Order of data")
  # horizontal line at zero
  abline(h = 0, col = "gray75")

plot(rsm.6.8.y.SOx1x2x3, which = c(1,4))

# Normality of Residuals
library(car)
qqPlot(rsm.6.8.y.SOx1x2x3$studres, las = 1, id.n = 3, main="QQ Plot")

##  6 12  3
##  1 27 26

cooks.distance(rsm.6.8.y.SOx1x2x3)

##         1         2         3         4         5         6         7
## 0.0002802 0.1126115 0.2626938 0.0356606 0.0318506 0.2451862 0.0059298
```

```
##         8         9        10        11        12        13        14
## 0.0014668 0.0243462 0.0156794 0.0179456 0.2110429 0.0116210 0.0032970
##        15        16        17        18        19        20        21
## 0.0684658 0.0464677 0.0004003 0.0236843 0.0144369 0.0545189 0.0003104
##        22        23        24        25        26        27
## 0.0489052 0.0360773 0.0713371 0.1607782 0.0160820 0.0122448
```



The Box-Cox transformation suggests a power transformation in the range for λ roughly from -0.5 to 0. That includes the log() transformation (for $\lambda = 0$).

```
library(MASS)
boxcox(rsm.6.8.y.SOx1x2x3, lambda = seq(-1, 2, by = 0.1))
```



Let's redo the analysis with $\log(y)$ as the response.

```
df.6.8$logy <- log(df.6.8$y)

library(rsm)
rsm.6.8.logy.SOx1x2x3 <- rsm(logy ~ SO(x1, x2, x3), data = df.6.8)
# externally Studentized residuals
rsm.6.8.logy.SOx1x2x3$studres <- rstudent(rsm.6.8.logy.SOx1x2x3)
summary(rsm.6.8.logy.SOx1x2x3)

##
## Call:
## rsm(formula = logy ~ SO(x1, x2, x3), data = df.6.8)
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   6.4156     0.1038   61.81  < 2e-16 ***
## x1            0.8248     0.0480   17.17  3.6e-12 ***
## x2           -0.6310     0.0480  -13.13  2.5e-10 ***
## x3           -0.3849     0.0480   -8.01  3.6e-07 ***
## x1:x2        -0.0382     0.0588   -0.65     0.52
## x1:x3        -0.0570     0.0588   -0.97     0.35
## x2:x3        -0.0208     0.0588   -0.35     0.73
## x1^2         -0.0933     0.0832   -1.12     0.28
## x2^2          0.0394     0.0832    0.47     0.64
## x3^2         -0.0750     0.0832   -0.90     0.38
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.969,Adjusted R-squared:  0.953
## F-statistic: 59.5 on 9 and 17 DF,  p-value: 4.44e-11
##
## Analysis of Variance Table
##
## Response: logy
##                 Df Sum Sq Mean Sq F value  Pr(>F)
## FO(x1, x2, x3)   3  22.08    7.36  177.11 5.1e-13
```

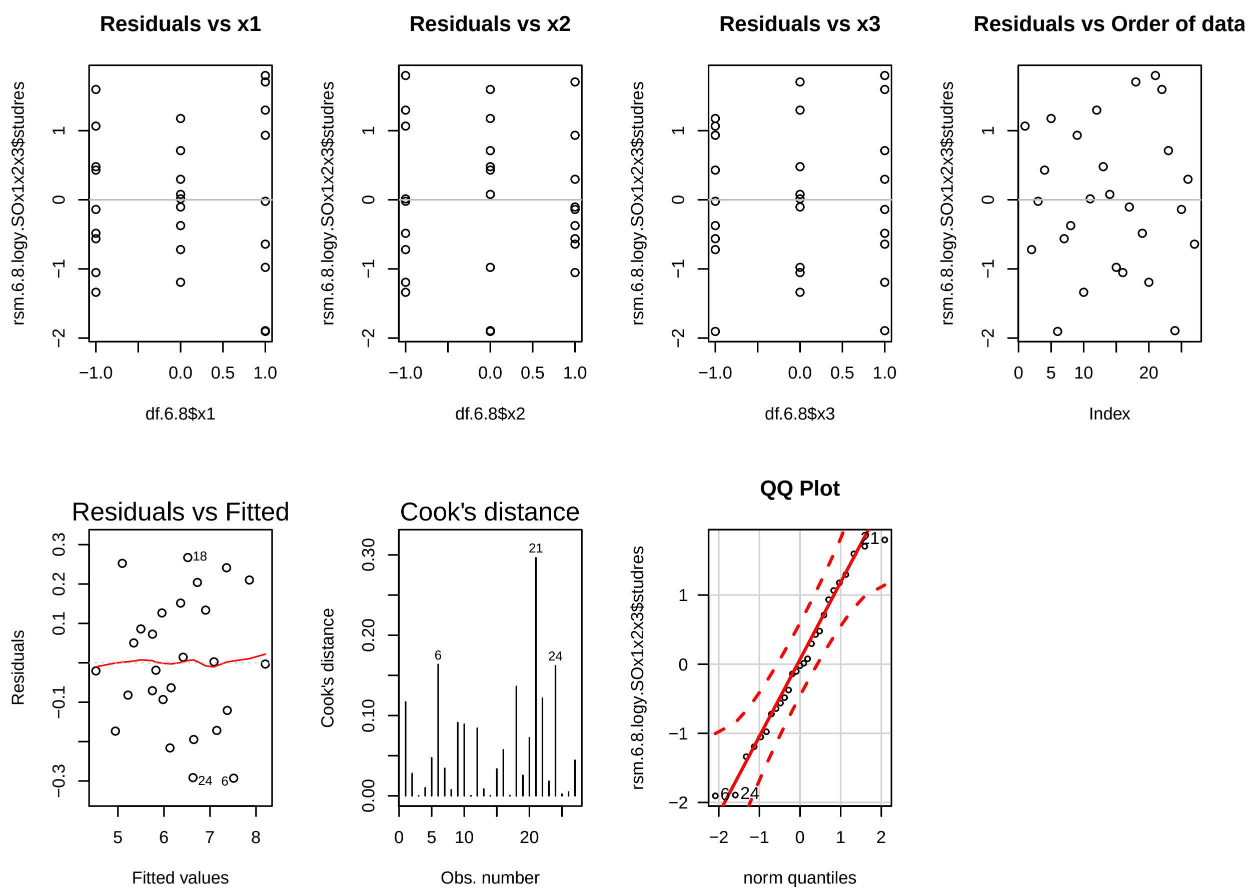
```
## TWI(x1, x2, x3)  3   0.06    0.02    0.50    0.69
## PQ(x1, x2, x3)   3   0.10    0.03    0.76    0.53
## Residuals       17   0.71    0.04
## Lack of fit     17   0.71    0.04
## Pure error       0   0.00
##
## Stationary point of response surface:
##     x1     x2     x3
##  4.296  8.669 -5.401
##
## Eigenanalysis:
## $values
## [1]  0.04244 -0.05429 -0.11708
##
## $vectors
##         [,1]     [,2]   [,3]
## x1  0.12756  0.58118 0.8037
## x2 -0.99020  0.02819 0.1368
## x3  0.05683 -0.81329 0.5791
```

This second-order model fits pretty well — and only main effects are significant! This greatly simplifies the model and interpretation.

The residual plots do not indicate a problem with normality of these residuals.

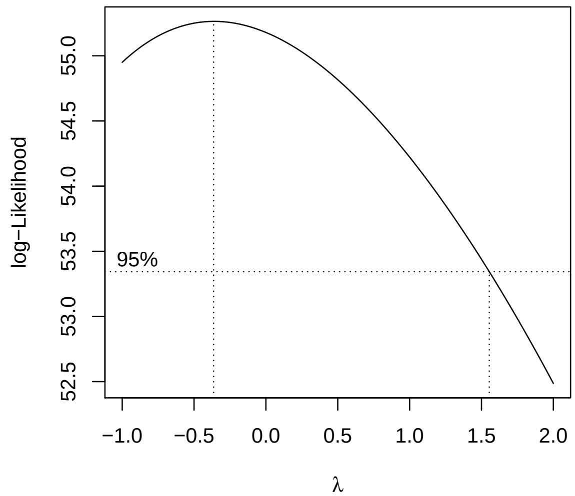
```
# plot diagnistics
par(mfrow=c(2,4))

plot(df.6.8$x1, rsm.6.8.logy.SOx1x2x3$studres, main="Residuals vs x1")
  # horizontal line at zero
  abline(h = 0, col = "gray75")

plot(df.6.8$x2, rsm.6.8.logy.SOx1x2x3$studres, main="Residuals vs x2")
  # horizontal line at zero
  abline(h = 0, col = "gray75")

plot(df.6.8$x3, rsm.6.8.logy.SOx1x2x3$studres, main="Residuals vs x3")
  # horizontal line at zero
  abline(h = 0, col = "gray75")

# residuals vs order of data
plot(rsm.6.8.logy.SOx1x2x3$studres, main="Residuals vs Order of data")
  # horizontal line at zero
  abline(h = 0, col = "gray75")

plot(rsm.6.8.logy.SOx1x2x3, which = c(1,4))

# Normality of Residuals
library(car)
qqPlot(rsm.6.8.logy.SOx1x2x3$studres, las = 1, id.n = 3, main="QQ Plot")

##  6 24 21
```

```
##  1  2 27
cooks.distance(rsm.6.8.logy.SOx1x2x3)
##           1           2           3           4           5           6           7
## 1.170e-01 2.784e-02 5.057e-05 1.009e-02 4.736e-02 1.637e-01 3.420e-02
##           8           9          10          11          12          13          14
## 7.657e-03 9.109e-02 8.910e-02 6.941e-06 8.428e-02 8.412e-03 2.278e-04
##          15          16          17          18          19          20          21
## 3.351e-02 5.738e-02 4.027e-04 1.362e-01 2.550e-02 7.237e-02 2.964e-01
##          22          23          24          25          26          27
## 1.217e-01 1.823e-02 1.621e-01 2.152e-03 4.857e-03 4.431e-02
```



Just to check, yes, for the Box-Cox transformation on $\log(y)$, the power $\lambda = 1$ is in the interval.

```
library(MASS)
boxcox(rsm.6.8.logy.SOx1x2x3, lambda = seq(-1, 2, by = 0.1))
```


log−Likelihood
55.0
54.5
54.0
53.5
53.0
52.5
95%
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
λ

Chapter 7

Experimental Designs for Fitting Response Surfaces — I

7.1 Example 7.6, Table 7.6, p. 332

```
#### 7.6
fn.data <- "http://statacumen.com/teach/RSM/data/RSM_EX_07-06.txt"
df.7.6 <- read.table(fn.data, header=TRUE)
df.7.6$block <- factor(df.7.6$block)
str(df.7.6)
```

```
## 'data.frame': 14 obs. of  4 variables:
##  $ x1   : num  -1 -1 1 1 0 0 0 0 0 0 ...
##  $ x2   : num  -1 1 -1 1 0 0 0 0 0 0 ...
##  $ block: Factor w/ 2 levels "1","2": 1 1 1 1 1 1 1 2 2 2 ...
##  $ y    : num  80.5 81.5 82 83.5 83.9 84.3 84 79.7 79.8 79.5 ...
```

```
df.7.6
```

```
##        x1     x2 block    y
## 1  -1.000 -1.000     1 80.5
## 2  -1.000  1.000     1 81.5
## 3   1.000 -1.000     1 82.0
## 4   1.000  1.000     1 83.5
## 5   0.000  0.000     1 83.9
## 6   0.000  0.000     1 84.3
## 7   0.000  0.000     1 84.0
## 8   0.000  0.000     2 79.7
## 9   0.000  0.000     2 79.8
## 10  0.000  0.000     2 79.5
## 11  1.414  0.000     2 78.4
## 12 -1.414  0.000     2 75.6
## 13  0.000  1.414     2 78.5
## 14  0.000 -1.414     2 77.0
```

Fit second-order linear model, without blocks.

```
library(rsm)
rsm.7.6.y.SOx1x2 <- rsm(y ~ SO(x1, x2), data = df.7.6)
# externally Studentized residuals
rsm.7.6.y.SOx1x2$studres <- rstudent(rsm.7.6.y.SOx1x2)
summary(rsm.7.6.y.SOx1x2)
```

```
##
## Call:
## rsm(formula = y ~ SO(x1, x2), data = df.7.6)
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   81.866      1.205   67.92  2.5e-12 ***
## x1             0.933      1.044    0.89     0.40
## x2             0.578      1.044    0.55     0.60
## x1:x2          0.125      1.476    0.08     0.93
## x1^2          -1.308      1.087   -1.20     0.26
## x2^2          -0.933      1.087   -0.86     0.42
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.283,Adjusted R-squared:  -0.166
```

```
## F-statistic: 0.63 on 5 and 8 DF,  p-value: 0.683
##
## Analysis of Variance Table
##
## Response: y
##              Df Sum Sq Mean Sq F value Pr(>F)
## FO(x1, x2)   2    9.6    4.81    0.55   0.60
## TWI(x1, x2)  1    0.1    0.06    0.01   0.93
## PQ(x1, x2)   2   17.8    8.89    1.02   0.40
## Residuals    8   69.7    8.72
## Lack of fit  3   40.6   13.52    2.32   0.19
## Pure error   5   29.2    5.83
##
## Stationary point of response surface:
##     x1     x2
## 0.3724 0.3345
##
## Eigenanalysis:
## $values
## [1] -0.9229 -1.3183
##
## $vectors
##        [,1]    [,2]
## x1 -0.1601 -0.9871
## x2 -0.9871  0.1601
```

Fit second-order linear model, with blocks.

```
library(rsm)
rsm.7.6.y.b.SOx1x2 <- rsm(y ~ block + SO(x1, x2), data = df.7.6)
# externally Studentized residuals
rsm.7.6.y.b.SOx1x2$studres <- rstudent(rsm.7.6.y.b.SOx1x2)
summary(rsm.7.6.y.b.SOx1x2)
```

```
##
## Call:
## rsm(formula = y ~ block + SO(x1, x2), data = df.7.6)
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  84.0954     0.0796 1056.07  < 2e-16 ***
## block2       -4.4575     0.0872  -51.10  2.9e-10 ***
## x1            0.9325     0.0577   16.16  8.4e-07 ***
## x2            0.5777     0.0577   10.01  2.1e-05 ***
## x1:x2         0.1250     0.0816    1.53     0.17
## x1^2         -1.3086     0.0601  -21.79  1.1e-07 ***
## x2^2         -0.9334     0.0601  -15.54  1.1e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.998,Adjusted R-squared:  0.996
## F-statistic:  607 on 6 and 7 DF,  p-value: 3.81e-09
##
## Analysis of Variance Table
##
```

```
## Response: y
##              Df Sum Sq Mean Sq F value  Pr(>F)
## block         1   69.5    69.5 2611.10 2.9e-10
## FO(x1, x2)    2    9.6     4.8  180.73 9.5e-07
## TWI(x1, x2)   1    0.1     0.1    2.35    0.17
## PQ(x1, x2)    2   17.8     8.9  334.05 1.1e-07
## Residuals     7    0.2     0.0
## Lack of fit   3    0.1     0.0    0.53    0.69
## Pure error    4    0.1     0.0
##
## Stationary point of response surface:
##     x1     x2
## 0.3723 0.3344
##
## Eigenanalysis:
## $values
## [1] -0.9233 -1.3187
##
## $vectors
##       [,1]    [,2]
## x1 -0.1601 -0.9871
## x2 -0.9871  0.1601
```

Fit second-order linear model, with blocks (as continuous and coded).

```
df.7.6$block.num <- as.numeric(df.7.6$block) - 1.5
df.7.6
```

```
##        x1     x2 block    y block.num
## 1  -1.000 -1.000     1 80.5      -0.5
## 2  -1.000  1.000     1 81.5      -0.5
## 3   1.000 -1.000     1 82.0      -0.5
## 4   1.000  1.000     1 83.5      -0.5
## 5   0.000  0.000     1 83.9      -0.5
## 6   0.000  0.000     1 84.3      -0.5
## 7   0.000  0.000     1 84.0      -0.5
## 8   0.000  0.000     2 79.7       0.5
## 9   0.000  0.000     2 79.8       0.5
## 10  0.000  0.000     2 79.5       0.5
## 11  1.414  0.000     2 78.4       0.5
## 12 -1.414  0.000     2 75.6       0.5
## 13  0.000  1.414     2 78.5       0.5
## 14  0.000 -1.414     2 77.0       0.5
```

```
library(rsm)
rsm.7.6.y.bn.SOx1x2 <- rsm(y ~ block.num + SO(x1, x2), data = df.7.6)
# externally Studentized residuals
rsm.7.6.y.bn.SOx1x2$studres <- rstudent(rsm.7.6.y.bn.SOx1x2)
summary(rsm.7.6.y.bn.SOx1x2)
```

```
##
## Call:
## rsm(formula = y ~ block.num + SO(x1, x2), data = df.7.6)
##
##              Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)  81.8667     0.0666 1228.86  < 2e-16 ***
## block.num    -4.4575     0.0872  -51.10  2.9e-10 ***
## x1            0.9325     0.0577   16.16  8.4e-07 ***
## x2            0.5777     0.0577   10.01  2.1e-05 ***
## x1:x2         0.1250     0.0816    1.53     0.17
## x1^2         -1.3086     0.0601  -21.79  1.1e-07 ***
## x2^2         -0.9334     0.0601  -15.54  1.1e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.998,Adjusted R-squared:  0.996
## F-statistic:  607 on 6 and 7 DF,  p-value: 3.81e-09
##
## Analysis of Variance Table
##
## Response: y
##             Df Sum Sq Mean Sq F value  Pr(>F)
## block.num    1   69.5    69.5 2611.10 2.9e-10
## FO(x1, x2)   2    9.6     4.8  180.73 9.5e-07
## TWI(x1, x2)  1    0.1     0.1    2.35    0.17
## PQ(x1, x2)   2   17.8     8.9  334.05 1.1e-07
## Residuals    7    0.2     0.0
## Lack of fit  3    0.1     0.0    0.53    0.69
## Pure error   4    0.1     0.0
##
## Stationary point of response surface:
##     x1     x2
## 0.3723 0.3344
##
## Eigenanalysis:
## $values
## [1] -0.9233 -1.3187
##
## $vectors
##       [,1]    [,2]
## x1 -0.1601 -0.9871
## x2 -0.9871  0.1601
```

Chapter 8

Experimental Designs for Fitting Response Surfaces — II

8.1 Evaluating design efficiencies

Here are some ideas for evaluating efficiencies of designs. Let's start with a single rep of a 2^3 design.

```
#### 8.1
df.8.1 <- read.table(text="
x1 x2 x3
-1 -1 -1
 1 -1 -1
-1  1 -1
 1  1 -1
-1 -1  1
 1 -1  1
-1  1  1
 1  1  1
", header=TRUE)
str(df.8.1)

## 'data.frame': 8 obs. of  3 variables:
##  $ x1: int  -1 1 -1 1 -1 1 -1 1
##  $ x2: int  -1 -1 1 1 -1 -1 1 1
##  $ x3: int  -1 -1 -1 -1 1 1 1 1
```

You can use the package `AlgDesign` function `eval.design()` to evaluate the D and A efficiencies of the design. The efficiency depends on the model function you specify.

```
library(AlgDesign)
eval.design(~ x1 + x2 + x3, df.8.1)

## $determinant
## [1] 1
##
## $A
## [1] 1
##
## $diagonality
## [1] 1
##
## $gmean.variances
## [1] 1

# these can be assigned to variables
D.eff.lin <-     eval.design(~ x1 + x2 + x3, df.8.1)$determinant
# note that A-efficiency is 1/A given from eval.design()
A.eff.lin <- 1 / eval.design(~ x1 + x2 + x3, df.8.1)$A
c(D.eff.lin, A.eff.lin)

## [1] 1 1
```

Now we calculate the same quantities after adding 3 center points.

```
df.8.1c <- rbind(df.8.1, data.frame(x1 = rep(0,3), x2 = rep(0,3), x3 = rep(0,3)))
df.8.1c
```

```
##    x1 x2 x3
## 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
## 9   0  0  0
## 10  0  0  0
## 11  0  0  0
```

```
library(AlgDesign)
eval.design(~ x1 + x2 + x3, df.8.1c)
```

```
## $determinant
## [1] 0.7875
##
## $A
## [1] 1.281
##
## $diagonality
## [1] 1
##
## $gmean.variances
## [1] 1.375
```

```
D.eff.lin <-     eval.design(~ x1 + x2 + x3, df.8.1c)$determinant
A.eff.lin <- 1 / eval.design(~ x1 + x2 + x3, df.8.1c)$A
c(D.eff.lin, A.eff.lin)
```

```
## [1] 0.7875 0.7805
```

8.2 Augmenting experimental designs

Sometimes it is desirable to augment a design in an optimal way. Suppose it is expected that a linear model will suffice for the experiment and so the design in `df.8.2` is run.

```
#### 8.2
df.8.2 <- read.table(text="
x1 x2
-1 -1
 1 -1
-1  1
 0  0
 0  0
", header=TRUE)
str(df.8.2)
```

```
## 'data.frame': 5 obs. of  2 variables:
##  $ x1: int  -1 1 -1 0 0
##  $ x2: int  -1 -1 1 0 0
```

Later, the experimenters realized that a second-order design would be more appropriate. They constructed a sensible list of design points to include (based on a face-centered CCD).

Then they evaluate the star points at a number of values (0.5, 1.0, 1.5, 2.0), and find the D-optimal augmentation at each.

```
for (i.alpha in c(0.5, 1.0, 1.5, 2.0)) {
  # augment design candidate points
  D.candidate.points <- i.alpha * data.frame(x1 = c( 1, -1,  0,  0)
                                            , x2 = c( 0,  0,  1, -1))

  # combine original points with candidate points
  D.all <- rbind(df.8.2, D.candidate.points)

  # keep the original points and augment the design with 4 points
  library(AlgDesign)
  D.aug <- optFederov(~ quad(.), D.all, nTrials = dim(df.8.2)[1] + 4, rows = 1:dim(df.8.2)[1]
                      , augment = TRUE, criterion = "D", maxIteration = 1e4, nRepeats = 1e2)
  ## same as:
  #D.aug <- optFederov(~ x1 + x2 + x1:x2 + x1^2 + x2^2
  #                    , D.all, nTrials = dim(df.8.2)[1] + 4, rows = 1:dim(df.8.2)[1]
  #                    , augment = TRUE, criterion = "D", maxIteration = 1e4, nRepeats = 1e2

  # Added points
  print(D.aug$design[(dim(df.8.2)[1]+1):dim(D.aug$design)[1],])

  library(AlgDesign)
  # quadratic
  #rm(D.eff.lin, A.eff.lin)
```

```
  D.eff.lin <-     eval.design(~ quad(.), D.aug$design)$determinant
  A.eff.lin <- 1 / eval.design(~ quad(.), D.aug$design)$A
  print(c(i.alpha, D.eff.lin, A.eff.lin))
}
```

```
##     x1   x2
## 6  0.5  0.0
## 7 -0.5  0.0
## 8  0.0  0.5
## 9  0.0 -0.5
## [1] 0.50000 0.18812 0.05287
##   x1 x2
## 6  1  0
## 7 -1  0
## 8  0  1
## 9  0 -1
## [1] 1.0000 0.3813 0.2556
##     x1   x2
## 6  1.5  0.0
## 7 -1.5  0.0
## 8  0.0  1.5
## 9  0.0 -1.5
## [1] 1.5000 0.6584 0.4209
##   x1 x2
## 6  2  0
## 7 -2  0
## 8  0  2
## 9  0 -2
## [1] 2.0000 1.1079 0.6066
```

Funding comes through and they don't have to just add the four axial points, but can afford to add any 12 points they like to the initial run. They look at a range of combinations of points for x_1 and x_2 and find the D-optimal augmentation.

```
# augment design candidate points
D.candidate.points <- expand.grid(x1 = seq(-2, 2, by = 0.5)
                                , x2 = seq(-2, 2, by = 0.5))
str(D.candidate.points)
```

```
## 'data.frame': 81 obs. of  2 variables:
##  $ x1: num  -2 -1.5 -1 -0.5 0 0.5 1 1.5 2 -2 ...
##  $ x2: num  -2 -2 -2 -2 -2 -2 -2 -2 -2 -1.5 ...
##  - attr(*, "out.attrs")=List of 2
##   ..$ dim     : Named int  9 9
##   .. ..- attr(*, "names")= chr  "x1" "x2"
##   ..$ dimnames:List of 2
##   .. ..$ x1: chr  "x1=-2.0" "x1=-1.5" "x1=-1.0" "x1=-0.5" ...
##   .. ..$ x2: chr  "x2=-2.0" "x2=-1.5" "x2=-1.0" "x2=-0.5" ...
```

```
head(D.candidate.points)
```

```
##     x1 x2
```

```
## 1 -2.0 -2
## 2 -1.5 -2
## 3 -1.0 -2
## 4 -0.5 -2
## 5  0.0 -2
## 6  0.5 -2
```

```
tail(D.candidate.points)
```

```
##      x1 x2
## 76 -0.5  2
## 77  0.0  2
## 78  0.5  2
## 79  1.0  2
## 80  1.5  2
## 81  2.0  2
```

```
# combine original points with candidate points
D.all <- rbind(df.8.2, D.candidate.points)

# keep the original points and augment the design with 4 points
library(AlgDesign)
D.aug <- optFederov(~ quad(.), D.all, nTrials = dim(df.8.2)[1] + 12, rows = 1:dim(df.8.2)[1]
                    , augment = TRUE, criterion = "D", maxIteration = 1e4, nRepeats = 1e2)

# Added points
print(D.aug$design[(dim(df.8.2)[1]+1):dim(D.aug$design)[1],])
```

```
##      x1   x2
## 6  -2.0 -2.0
## 7  -1.5 -2.0
## 13  1.5 -2.0
## 14  2.0 -2.0
## 42 -2.0  0.0
## 50  2.0  0.0
## 69 -2.0  1.5
## 77  2.0  1.5
## 78 -2.0  2.0
## 82  0.0  2.0
## 83  0.5  2.0
## 86  2.0  2.0
```

```
library(AlgDesign)
# quadratic
#rm(D.eff.lin, A.eff.lin)
D.eff.lin <-     eval.design(~ quad(.), D.aug$design)$determinant
A.eff.lin <- 1 / eval.design(~ quad(.), D.aug$design)$A
print(c(i.alpha, D.eff.lin, A.eff.lin))
```

```
## [1] 2.000 2.517 1.026
```

```
# full design
library(plyr)
D.aug.ordered <- arrange(D.aug$design, x1, x2)
D.aug.ordered
```

```
##      x1   x2
```

```
## 1  -2.0 -2.0
## 2  -2.0  0.0
## 3  -2.0  1.5
## 4  -2.0  2.0
## 5  -1.5 -2.0
## 6  -1.0 -1.0
## 7  -1.0  1.0
## 8   0.0  0.0
## 9   0.0  0.0
## 10  0.0  2.0
## 11  0.5  2.0
## 12  1.0 -1.0
## 13  1.5 -2.0
## 14  2.0 -2.0
## 15  2.0  0.0
## 16  2.0  1.5
## 17  2.0  2.0
```

8.3 Example 8.10, p. 390

The `AlgDesign` package can be used to create computer-generated designs, providing a list of permitted/suggested design points.

D-optimal $3 \times 2 \times 2$ for $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_{11} x_1^2 + \varepsilon$.

```
#### 8.10
# design candidate values  3 * 2 * 2
D.candidate.points <- expand.grid(x1 = seq(-1, 1, by = 1)
                                , x2 = c(-1, 1)
                                , x3 = c(-1, 1))
D.candidate.points
```

```
##    x1 x2 x3
## 1  -1 -1 -1
## 2   0 -1 -1
## 3   1 -1 -1
## 4  -1  1 -1
## 5   0  1 -1
## 6   1  1 -1
## 7  -1 -1  1
## 8   0 -1  1
## 9   1 -1  1
## 10 -1  1  1
## 11  0  1  1
## 12  1  1  1
```

```
# choose the 12 points based on D-criterion
library(AlgDesign)
D.gen <- optFederov(~ x1 + x2 + x3 + x1^2, D.candidate.points, nTrials = 12
                    , criterion = "D", evaluateI = TRUE, maxIteration = 1e4, nRepeats = 1e2)
D.gen
```

```
## $D
## [1] 0.9036
##
## $A
## [1] 1.125
##
## $I
## [1] 4
##
## $Ge
## [1] 0.889
##
## $Dea
## [1] 0.882
##
## $design
##    x1 x2 x3
## 1  -1 -1 -1
## 2   0 -1 -1
## 3   1 -1 -1
```

```
## 4  -1  1 -1
## 5   0  1 -1
## 6   1  1 -1
## 7  -1 -1  1
## 8   0 -1  1
## 9   1 -1  1
## 10 -1  1  1
## 11  0  1  1
## 12  1  1  1
##
## $rows
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

```
A.eff <- 1/D.gen$A # A efficiency is the reciprocal of $A

# easier to read when ordered
library(plyr)
D.gen.ordered <- arrange(D.gen$design, x1, x2, x3)
D.gen.ordered
```

```
##    x1 x2 x3
## 1  -1 -1 -1
## 2  -1 -1  1
## 3  -1  1 -1
## 4  -1  1  1
## 5   0 -1 -1
## 6   0 -1  1
## 7   0  1 -1
## 8   0  1  1
## 9   1 -1 -1
## 10  1 -1  1
## 11  1  1 -1
## 12  1  1  1
```

D-optimal $5 \times 2 \times 2$ for $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_{11} x_1^2 + \varepsilon$.

```
#### 8.10
# design candidate values  5 * 2 * 2
D.candidate.points <- expand.grid(x1 = seq(-1, 1, by = 0.5)
                                , x2 = c(-1, 1)
                                , x3 = c(-1, 1))
D.candidate.points
```

```
##      x1 x2 x3
## 1  -1.0 -1 -1
## 2  -0.5 -1 -1
## 3   0.0 -1 -1
## 4   0.5 -1 -1
## 5   1.0 -1 -1
## 6  -1.0  1 -1
## 7  -0.5  1 -1
## 8   0.0  1 -1
## 9   0.5  1 -1
## 10  1.0  1 -1
## 11 -1.0 -1  1
```

```
## 12 -0.5 -1  1
## 13  0.0 -1  1
## 14  0.5 -1  1
## 15  1.0 -1  1
## 16 -1.0  1  1
## 17 -0.5  1  1
## 18  0.0  1  1
## 19  0.5  1  1
## 20  1.0  1  1
# choose the 12 points based on D-criterion
library(AlgDesign)
D.gen <- optFederov(~ x1 + x2 + x3 + x1^2, D.candidate.points, nTrials = 12
                    , criterion = "D", evaluateI = TRUE, maxIteration = 1e4, nRepeats = 1e2)
D.gen
## $D
## [1] 0.9306
##
## $A
## [1] 1.083
##
## $I
## [1] 3.667
##
## $Ge
## [1] 0.923
##
## $Dea
## [1] 0.92
##
## $design
##      x1 x2 x3
## 1  -1.0 -1 -1
## 4   0.5 -1 -1
## 5   1.0 -1 -1
## 6  -1.0  1 -1
## 7  -0.5  1 -1
## 10  1.0  1 -1
## 11 -1.0 -1  1
## 12 -0.5 -1  1
## 15  1.0 -1  1
## 16 -1.0  1  1
## 19  0.5  1  1
## 20  1.0  1  1
##
## $rows
##  [1]  1  4  5  6  7 10 11 12 15 16 19 20
A.eff <- 1/D.gen$A # A efficiency is the reciprocal of £A

# easier to read when ordered
library(plyr)
D.gen.ordered <- arrange(D.gen$design, x1, x2, x3)
```

```
D.gen.ordered
```

```
##      x1 x2 x3
## 1  -1.0 -1 -1
## 2  -1.0 -1  1
## 3  -1.0  1 -1
## 4  -1.0  1  1
## 5  -0.5 -1  1
## 6  -0.5  1 -1
## 7   0.5 -1 -1
## 8   0.5  1  1
## 9   1.0 -1 -1
## 10  1.0 -1  1
## 11  1.0  1 -1
## 12  1.0  1  1
```

I-optimal $5 \times 2 \times 2$ for $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_{11} x_1^2 + \varepsilon$.

```
#### 8.10
# design candidate values  5 * 2 * 2
D.candidate.points <- expand.grid(x1 = seq(-1, 1, by = 0.5)
                                , x2 = c(-1, 1)
                                , x3 = c(-1, 1))
D.candidate.points
```

```
##      x1 x2 x3
## 1  -1.0 -1 -1
## 2  -0.5 -1 -1
## 3   0.0 -1 -1
## 4   0.5 -1 -1
## 5   1.0 -1 -1
## 6  -1.0  1 -1
## 7  -0.5  1 -1
## 8   0.0  1 -1
## 9   0.5  1 -1
## 10  1.0  1 -1
## 11 -1.0 -1  1
## 12 -0.5 -1  1
## 13  0.0 -1  1
## 14  0.5 -1  1
## 15  1.0 -1  1
## 16 -1.0  1  1
## 17 -0.5  1  1
## 18  0.0  1  1
## 19  0.5  1  1
## 20  1.0  1  1
```

```
# choose the 12 points based on D-criterion
library(AlgDesign)
D.gen <- optFederov(~ x1 + x2 + x3 + x1^2, D.candidate.points, nTrials = 12
                    , criterion = "I", evaluateI = TRUE, maxIteration = 1e4, nRepeats = 1e2)
D.gen
```

```
## $D
## [1] 0.9306
##
```

```
## $A
## [1] 1.083
##
## $I
## [1] 3.667
##
## $Ge
## [1] 0.923
##
## $Dea
## [1] 0.92
##
## $design
##      x1 x2 x3
## 1  -1.0 -1 -1
## 4   0.5 -1 -1
## 5   1.0 -1 -1
## 6  -1.0  1 -1
## 7  -0.5  1 -1
## 10  1.0  1 -1
## 11 -1.0 -1  1
## 12 -0.5 -1  1
## 15  1.0 -1  1
## 16 -1.0  1  1
## 19  0.5  1  1
## 20  1.0  1  1
##
## $rows
##  [1]  1  4  5  6  7 10 11 12 15 16 19 20
A.eff <- 1/D.gen$A # A efficiency is the reciprocal of $A

# easier to read when ordered
library(plyr)
D.gen.ordered <- arrange(D.gen$design, x1, x2, x3)
D.gen.ordered
##      x1 x2 x3
## 1  -1.0 -1 -1
## 2  -1.0 -1  1
## 3  -1.0  1 -1
## 4  -1.0  1  1
## 5  -0.5 -1  1
## 6  -0.5  1 -1
## 7   0.5 -1 -1
## 8   0.5  1  1
## 9   1.0 -1 -1
## 10  1.0 -1  1
## 11  1.0  1 -1
## 12  1.0  1  1
```

A-optimal $5 \times 2 \times 2$ for $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_{11} x_1^2 + \varepsilon$.

```
#### 8.10
# design candidate values
D.candidate.points <- expand.grid(x1 = seq(-1, 1, by = 0.5)
                                , x2 = c(-1, 1)
                                , x3 = c(-1, 1))
#D.candidate.points

# choose the 12 points based on A-criterion
library(AlgDesign)
D.gen <- optFederov(~ x1 + x2 + x3 + x1^2, D.candidate.points, nTrials = 12
                    , criterion = "A", evaluateI = TRUE, maxIteration = 1e4, nRepeats = 1e2)
D.gen
```

```
## $D
## [1] 0.9306
##
## $A
## [1] 1.083
##
## $I
## [1] 3.667
##
## $Ge
## [1] 0.923
##
## $Dea
## [1] 0.92
##
## $design
##      x1 x2 x3
## 1  -1.0 -1 -1
## 2  -0.5 -1 -1
## 5   1.0 -1 -1
## 6  -1.0  1 -1
## 9   0.5  1 -1
## 10  1.0  1 -1
## 11 -1.0 -1  1
## 14  0.5 -1  1
## 15  1.0 -1  1
## 16 -1.0  1  1
## 17 -0.5  1  1
## 20  1.0  1  1
##
## $rows
##  [1]  1  2  5  6  9 10 11 14 15 16 17 20
```

```
A.eff <- 1/D.gen$A # A efficiency is the reciprocal of £A

# easier to read when ordered
library(plyr)
D.gen.ordered <- arrange(D.gen$design, x1, x2, x3)
D.gen.ordered
```

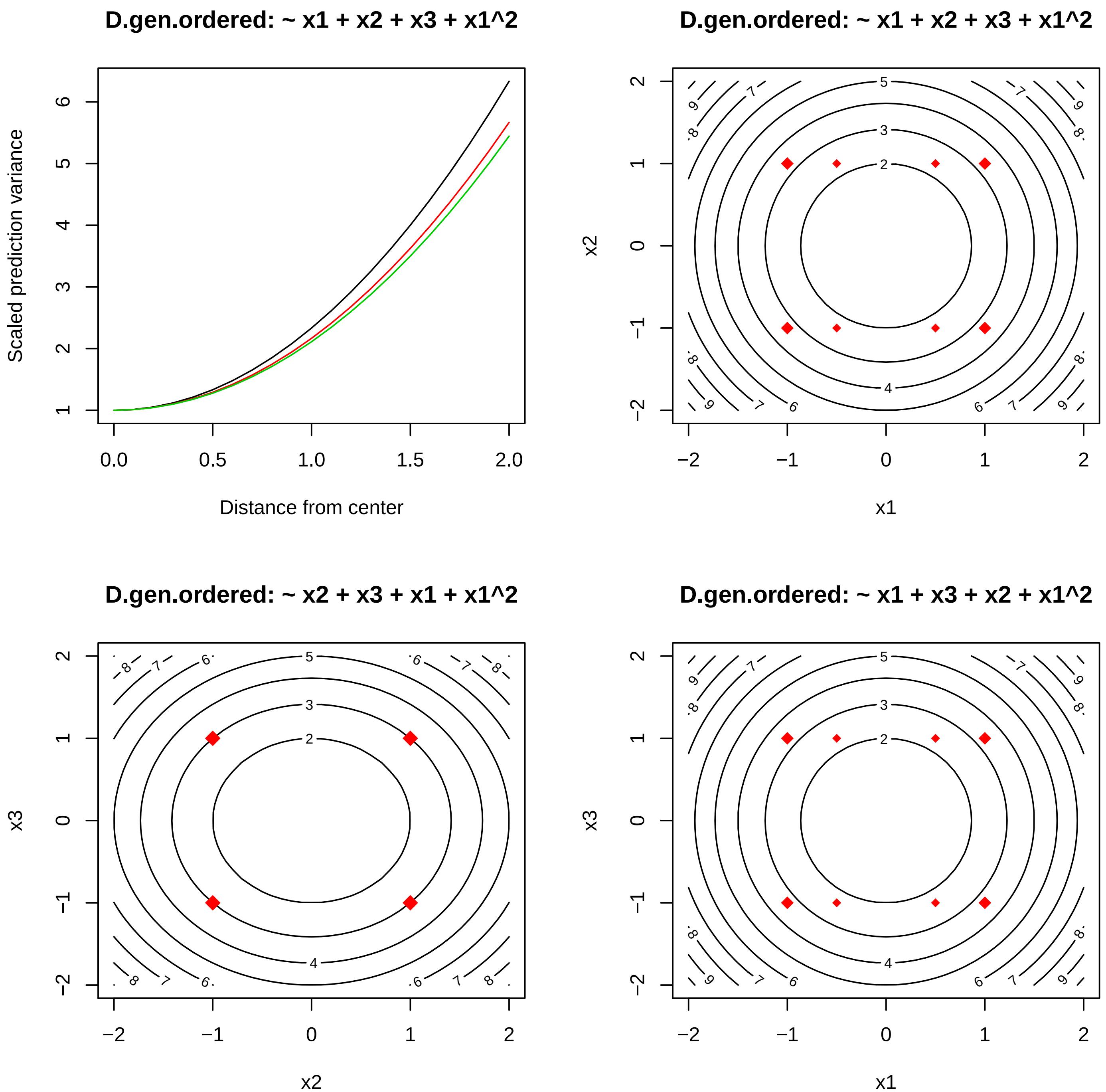
```
##      x1 x2 x3
```

```
## 1  -1.0 -1 -1
## 2  -1.0 -1  1
## 3  -1.0  1 -1
## 4  -1.0  1  1
## 5  -0.5 -1 -1
## 6  -0.5  1  1
## 7   0.5 -1  1
## 8   0.5  1 -1
## 9   1.0 -1 -1
## 10  1.0 -1  1
## 11  1.0  1 -1
## 12  1.0  1  1
```

All of these choose the same designs.

The function `varfcn()` computes the scaled variance function for a design, based on a specified model, and plots it either as a function of the radius in a specified direction or as a contour plot.

```
library(rsm)
par(mfrow = c(2,2))
varfcn(D.gen.ordered, ~ x1 + x2 + x3 + x1^2)
# the contour plots will plot the first two variables in the formula
varfcn(D.gen.ordered, ~ x1 + x2 + x3 + x1^2, contour = TRUE)
varfcn(D.gen.ordered, ~ x2 + x3 + x1 + x1^2, contour = TRUE)
varfcn(D.gen.ordered, ~ x1 + x3 + x2 + x1^2, contour = TRUE)
```


D.gen.ordered: ~ x1 + x2 + x3 + x1^2
Scaled prediction variance
Distance from center
D.gen.ordered: ~ x1 + x2 + x3 + x1^2
x2
x1
D.gen.ordered: ~ x2 + x3 + x1 + x1^2
x3
x2
D.gen.ordered: ~ x1 + x3 + x2 + x1^2
x3
x1

Chapter 9

Advanced Topics in Response Surface Methodology

(not covered)

Chapter 10

Robust Parameter Design and Process Robustness Studies

10.1 Example 10.5, p. 511

10.1.1 Model mean and ln of variance separately

Read data and calculate mean, variance, log-variance, and SNR.

```
#### 10.5
fn.data <- "http://statacumen.com/teach/RSM/data/RSM_EX_10-05.txt"
df.10.5 <- read.table(fn.data, header=TRUE)
df.10.5

##   x1 x2    y1    y2      y3    y4
## 1 -1 -1 33.50 41.23 25.2683 31.99
## 2 -1  0 35.82 38.07 32.7928 34.04
## 3 -1  1 33.08 31.84 36.2500 34.02
## 4  0 -1 30.45 41.29 15.1493 23.99
## 5  0  0 34.87 40.23 27.7724 31.13
## 6  0  1 35.22 37.10 33.3280 35.21
## 7  1 -1 21.16 34.11  0.7917 15.74
## 8  1  0 27.67 38.15 15.5132 25.99
## 9  1  1 32.12 38.12 26.1673 32.16

# calculate some summary statistics, especially SNR
df.10.5a       <- df.10.5
df.10.5a$m     <- apply(df.10.5a[,c("y1","y2","y3","y4")], 1, mean)
df.10.5a$s     <- apply(df.10.5a[,c("y1","y2","y3","y4")], 1, sd)
df.10.5a$s2    <- apply(df.10.5a[,c("y1","y2","y3","y4")], 1, var)
df.10.5a$logs2 <- log(df.10.5a$s2)
df.10.5a$SNR   <- apply(df.10.5a[,c("y1","y2","y3","y4")], 1
                    , function(x) {
                      -10 * log10( sum(1 / x^2) / length(x) )
                    })

# average SNR over each condition, independently
library(plyr)
df.10.5a.SNR <- rbind(
  ddply(df.10.5a, .(x1), function(.X) {
    data.frame(
        var   = "x1"
      , level = .X$x1[1]
      , m.SNR = mean(.X$SNR)
      , s.SNR = sd(.X$SNR)
      , min.SNR = min(.X$SNR)
      , max.SNR = max(.X$SNR)
      )
    })[,-1]
    ,
  ddply(df.10.5a, .(x2), function(.X) {
    data.frame(
        var   = "x2"
      , level = .X$x2[1]
      , m.SNR = mean(.X$SNR)
      , s.SNR = sd(.X$SNR)
```

```
      , min.SNR = min(.X$SNR)
      , max.SNR = max(.X$SNR)
      )
    })[,-1]
  )
df.10.5a.SNR
```

```
##   var level m.SNR   s.SNR min.SNR max.SNR
## 1  x1    -1 30.47  0.4600  29.976   30.89
## 2  x1     0 29.43  2.0275  27.122   30.92
## 3  x1     1 20.36 14.2561   3.972   29.91
## 4  x2    -1 20.36 14.2607   3.972   29.98
## 5  x2     0 29.45  1.9742  27.196   30.89
## 6  x2     1 30.46  0.5092  29.909   30.92
```

First, model mean and ln of variance separately, as suggested on p. 519 of text.

Fit second-order linear model for $\bar{y}$ and $\log(s^2)$.

```
library(rsm)
rsm.10.5a.m.SOx1x2 <- rsm(m ~ SO(x1, x2), data = df.10.5a)
# externally Studentized residuals
rsm.10.5a.m.SOx1x2$studres <- rstudent(rsm.10.5a.m.SOx1x2)
summary(rsm.10.5a.m.SOx1x2)
```

```
##
## Call:
## rsm(formula = m ~ SO(x1, x2), data = df.10.5a)
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  33.3889     0.0717   465.5  2.2e-08 ***
## x1           -4.1752     0.0393  -106.3  1.8e-06 ***
## x2            3.7481     0.0393    95.4  2.5e-06 ***
## x1:x2         3.3485     0.0481    69.6  6.5e-06 ***
## x1^2         -2.3277     0.0680   -34.2  5.5e-05 ***
## x2^2         -1.8670     0.0680   -27.4  0.00011 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:     1,Adjusted R-squared:     1
## F-statistic: 5.43e+03 on 5 and 3 DF,  p-value: 3.94e-06
##
## Analysis of Variance Table
##
## Response: m
##             Df Sum Sq Mean Sq F value  Pr(>F)
## FO(x1, x2)   2  188.9    94.4   10198 1.8e-06
## TWI(x1, x2)  1   44.8    44.8    4843 6.5e-06
## PQ(x1, x2)   2   17.8     8.9     961 6.1e-05
## Residuals    3    0.0     0.0
## Lack of fit  3    0.0     0.0
## Pure error   0    0.0
##
```

```
## Stationary point of response surface:
##      x1      x2
## -0.4926  0.5620
##
## Eigenanalysis:
## $values
## [1] -0.4073 -3.7874
##
## $vectors
##        [,1]    [,2]
## x1 -0.6572 -0.7538
## x2 -0.7538  0.6572
```

```
library(rsm)
rsm.10.5a.logs2.SOx1x2 <- rsm(logs2 ~ SO(x1, x2), data = df.10.5a)
# externally Studentized residuals
rsm.10.5a.logs2.SOx1x2$studres <- rstudent(rsm.10.5a.logs2.SOx1x2)
summary(rsm.10.5a.logs2.SOx1x2)
```

```
##
## Call:
## rsm(formula = logs2 ~ SO(x1, x2), data = df.10.5a)
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.9863     0.5415    5.51    0.012 *
## x1            1.0350     0.2966    3.49    0.040 *
## x2           -1.4212     0.2966   -4.79    0.017 *
## x1:x2         0.1095     0.3633    0.30    0.783
## x1^2          0.2516     0.5137    0.49    0.658
## x2^2          0.0262     0.5137    0.05    0.963
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.922,Adjusted R-squared:  0.792
## F-statistic: 7.09 on 5 and 3 DF,  p-value: 0.0688
##
## Analysis of Variance Table
##
## Response: logs2
##             Df Sum Sq Mean Sq F value Pr(>F)
## FO(x1, x2)   2  18.55    9.27   17.57  0.022
## TWI(x1, x2)  1   0.05    0.05    0.09  0.783
## PQ(x1, x2)   2   0.13    0.06    0.12  0.890
## Residuals    3   1.58    0.53
## Lack of fit  3   1.58    0.53
## Pure error   0   0.00
##
## Stationary point of response surface:
##     x1     x2
## -14.60  57.68
##
## Eigenanalysis:
## $values
```

```
## [1] 0.26416 0.01359
##
## $vectors
##         [,1]    [,2]
## x1 -0.9746  0.2241
## x2 -0.2241 -0.9746
```

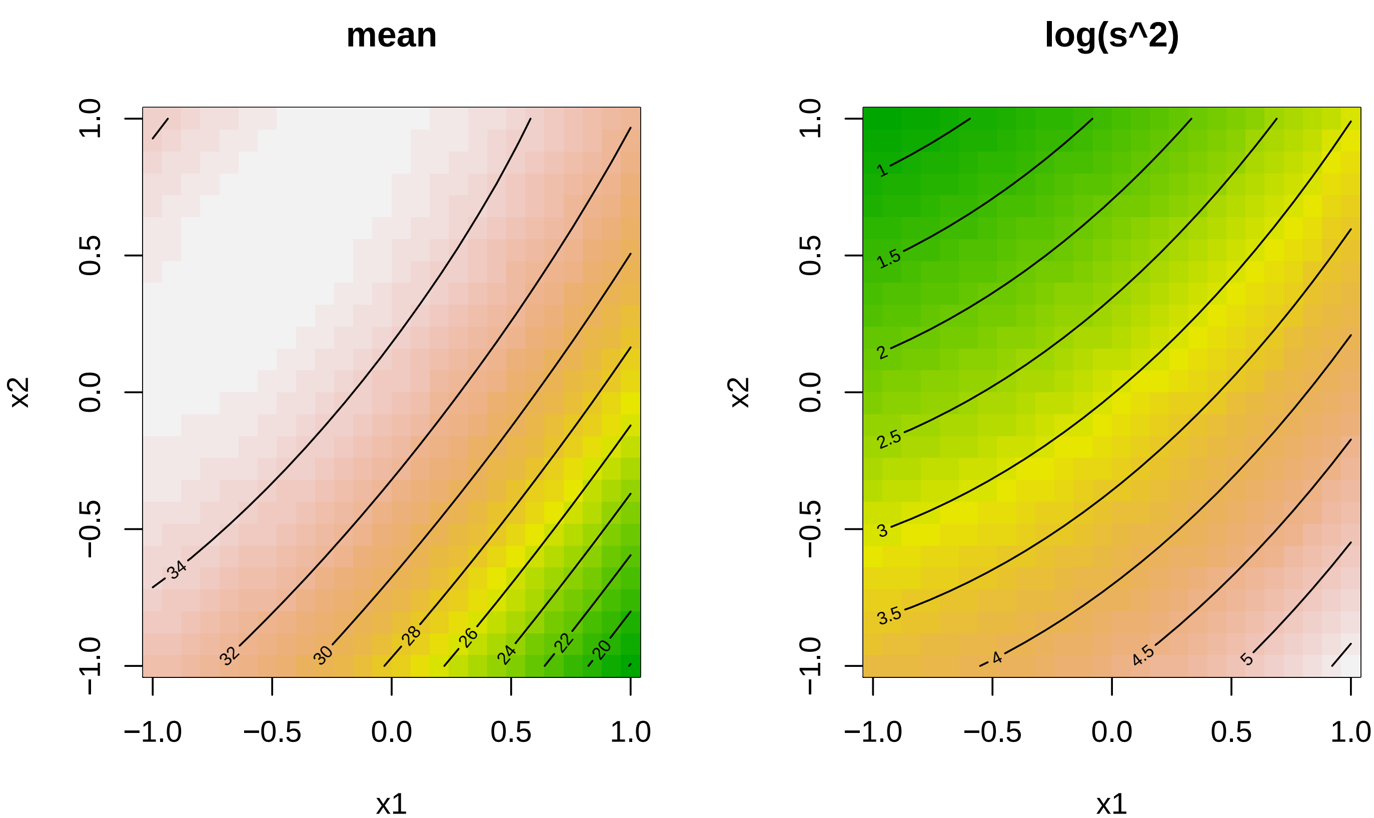
```
par(mfrow = c(1,2))
# this is the stationary point
canonical(rsm.10.5a.m.SOx1x2)$xs
```

```
##      x1      x2
## -0.4926  0.5620
```

```
contour(rsm.10.5a.m.SOx1x2, ~ x1 + x2, image=TRUE
      , at = canonical(rsm.10.5a.m.SOx1x2)$xs, main = "mean")
# this is the stationary point
canonical(rsm.10.5a.logs2.SOx1x2)$xs
```

```
##     x1     x2
## -14.60  57.68
```

```
contour(rsm.10.5a.logs2.SOx1x2, ~ x1 + x2, image=TRUE
      , at = canonical(rsm.10.5a.logs2.SOx1x2)$xs, main = "log(s^2)")
```



10.1.2 Model mean and ln of variance as on p. 512 (eq. 10.20)

Reshape data and create z_1 and z_2 variables.

```
#### 10.5b
fn.data <- "http://statacumen.com/teach/RSM/data/RSM_EX_10-05.txt"
df.10.5b <- read.table(fn.data, header=TRUE)
df.10.5b
```

```
##   x1 x2    y1    y2      y3    y4
## 1 -1 -1 33.50 41.23 25.2683 31.99
## 2 -1  0 35.82 38.07 32.7928 34.04
## 3 -1  1 33.08 31.84 36.2500 34.02
## 4  0 -1 30.45 41.29 15.1493 23.99
## 5  0  0 34.87 40.23 27.7724 31.13
## 6  0  1 35.22 37.10 33.3280 35.21
## 7  1 -1 21.16 34.11  0.7917 15.74
## 8  1  0 27.67 38.15 15.5132 25.99
## 9  1  1 32.12 38.12 26.1673 32.16
```

```
# reshape data into long format
library(reshape2)
df.10.5b.long <- melt(df.10.5b, id.vars = c("x1", "x2"), variable.name = "rep", value.name = "

# create z1 and z2 variables
df.10.5b.long$z1 <- NA
df.10.5b.long$z2 <- NA
df.10.5b.long[(df.10.5b.long$rep == "y1"), c("z1", "z2")] <- data.frame(z1 = -1, z2 = -1)
df.10.5b.long[(df.10.5b.long$rep == "y2"), c("z1", "z2")] <- data.frame(z1 = -1, z2 =  1)
df.10.5b.long[(df.10.5b.long$rep == "y3"), c("z1", "z2")] <- data.frame(z1 =  1, z2 = -1)
df.10.5b.long[(df.10.5b.long$rep == "y4"), c("z1", "z2")] <- data.frame(z1 =  1, z2 =  1)
df.10.5b.long
```

```
##    x1 x2 rep       y z1 z2
## 1  -1 -1  y1 33.5021 -1 -1
## 2  -1  0  y1 35.8234 -1 -1
## 3  -1  1  y1 33.0773 -1 -1
## 4   0 -1  y1 30.4481 -1 -1
## 5   0  0  y1 34.8679 -1 -1
## 6   0  1  y1 35.2202 -1 -1
## 7   1 -1  y1 21.1553 -1 -1
## 8   1  0  y1 27.6736 -1 -1
## 9   1  1  y1 32.1245 -1 -1
## 10 -1 -1  y2 41.2268 -1  1
## 11 -1  0  y2 38.0689 -1  1
## 12 -1  1  y2 31.8435 -1  1
## 13  0 -1  y2 41.2870 -1  1
## 14  0  0  y2 40.2276 -1  1
## 15  0  1  y2 37.1008 -1  1
## 16  1 -1  y2 34.1086 -1  1
## 17  1  0  y2 38.1477 -1  1
## 18  1  1  y2 38.1193 -1  1
## 19 -1 -1  y3 25.2683  1 -1
```

```
## 20 -1  0  y3 32.7928  1 -1
## 21 -1  1  y3 36.2500  1 -1
## 22  0 -1  y3 15.1493  1 -1
## 23  0  0  y3 27.7724  1 -1
## 24  0  1  y3 33.3280  1 -1
## 25  1 -1  y3  0.7917  1 -1
## 26  1  0  y3 15.5132  1 -1
## 27  1  1  y3 26.1673  1 -1
## 28 -1 -1  y4 31.9930  1  1
## 29 -1  0  y4 34.0383  1  1
## 30 -1  1  y4 34.0162  1  1
## 31  0 -1  y4 23.9883  1  1
## 32  0  0  y4 31.1321  1  1
## 33  0  1  y4 35.2085  1  1
## 34  1 -1  y4 15.7450  1  1
## 35  1  0  y4 25.9873  1  1
## 36  1  1  y4 32.1622  1  1
```

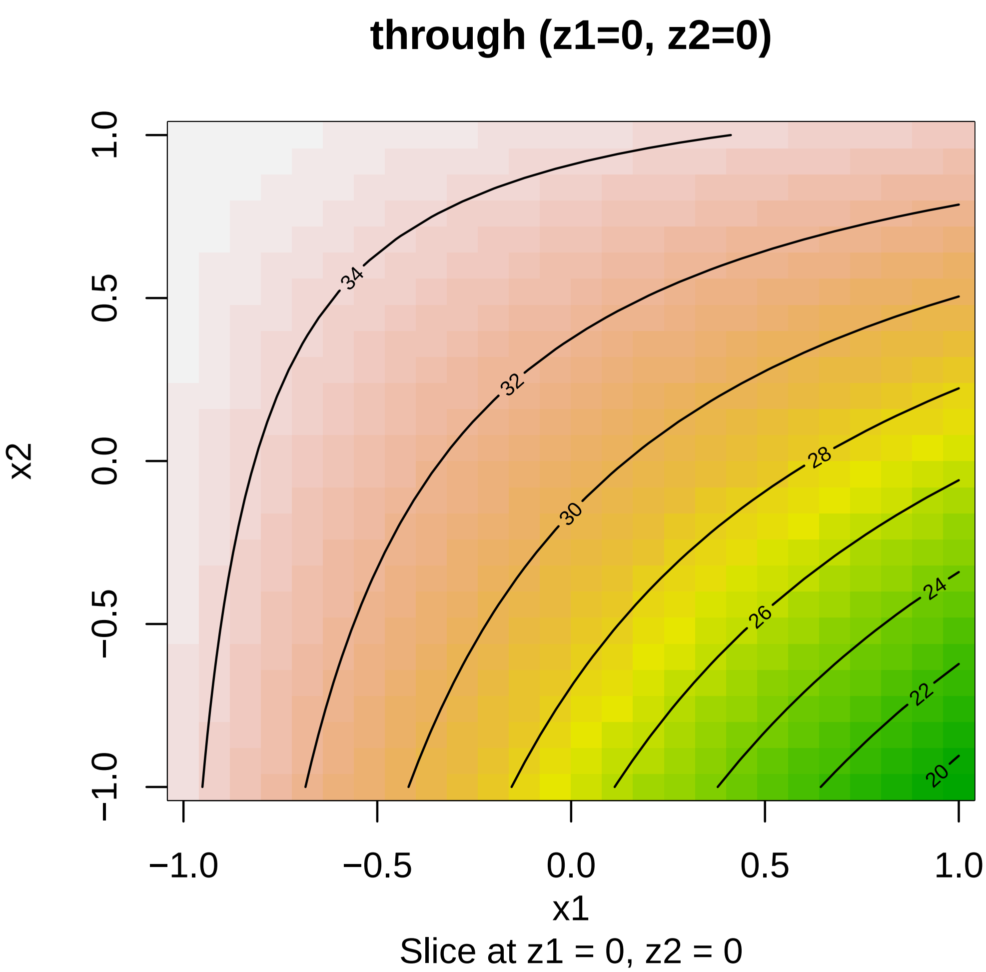
Model response using Equation 10.20 on p. 512.

```
library(rsm)
#rsm.10.5b.y.SPECx1x2z1z2 <- rsm(y ~ SO(x1, x2) + FO(z1, z2)
#                               + x1:z1 + x1:z2 + x2:z1 + x2:z2, data = df.10.5b.long)
lm.10.5b.y.SPECx1x2z1z2 <- lm(y ~ x1 + x2 + x1:x2 + x1:x1 + x2:x2
                                 + x1:x2 + x1:z1 + x1:z2 + x2:z1 + x2:z2
                              , data = df.10.5b.long)
# externally Studentized residuals
lm.10.5b.y.SPECx1x2z1z2$studres <- rstudent(lm.10.5b.y.SPECx1x2z1z2)
summary(lm.10.5b.y.SPECx1x2z1z2)
```

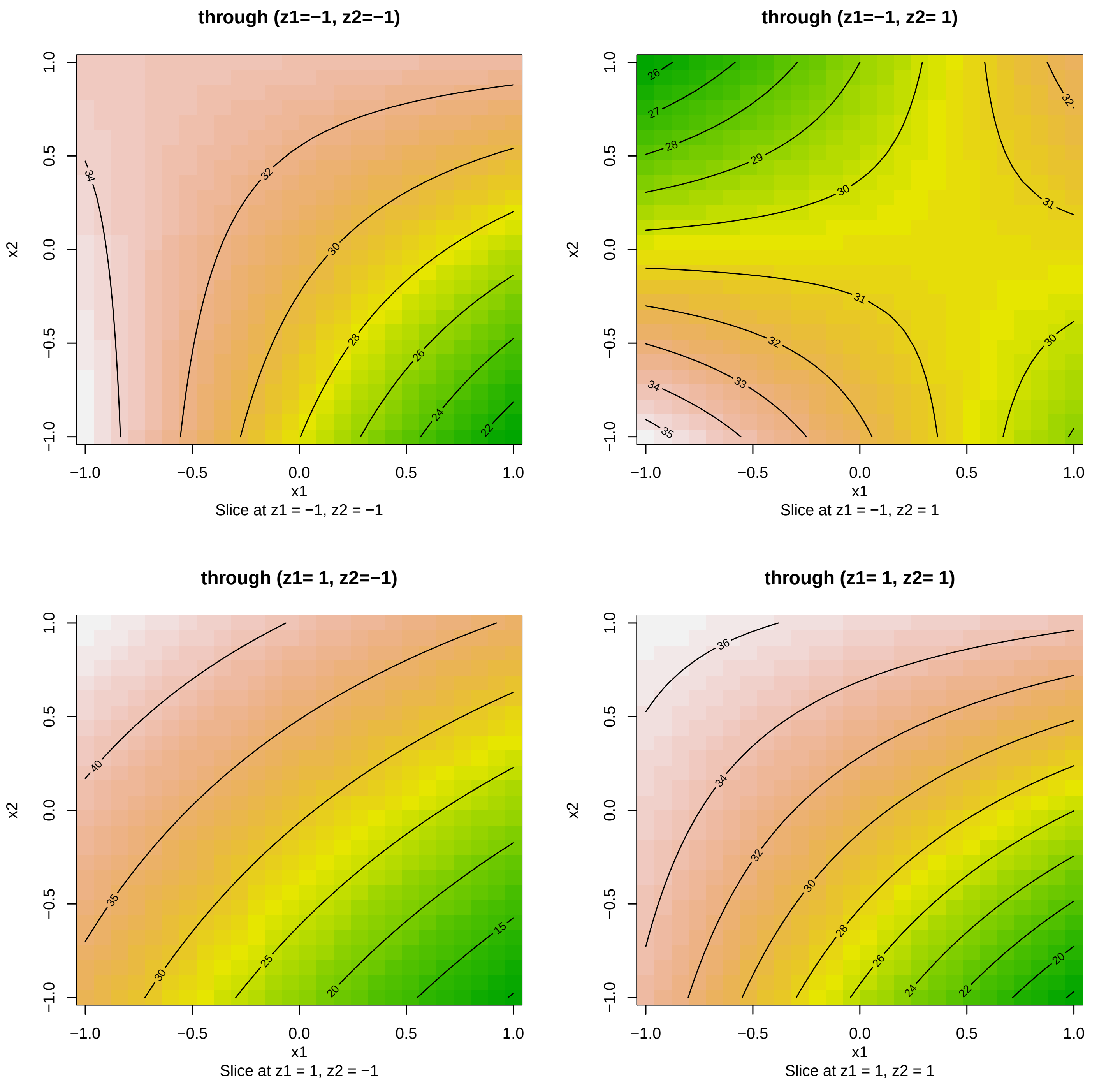
```
##
## Call:
## lm.default(formula = y ~ x1 + x2 + x1:x2 + x1:x1 + x2:x2 + x1:x2 +
##     x1:z1 + x1:z2 + x2:z1 + x2:z2, data = df.10.5b.long)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -8.931 -2.759 -0.415  4.504  9.635
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   30.592      0.998   30.66   <2e-16 ***
## x1            -4.175      1.222   -3.42   0.0020 **
## x2             3.748      1.222    3.07   0.0048 **
## x1:x2          3.348      1.496    2.24   0.0334 *
## x1:z1         -2.324      1.222   -1.90   0.0675 .
## x1:z2          1.932      1.222    1.58   0.1250
## x2:z1          3.268      1.222    2.67   0.0123 *
## x2:z2         -2.073      1.222   -1.70   0.1009
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.99 on 28 degrees of freedom
```

```
## Multiple R-squared:  0.601,Adjusted R-squared:  0.502
## F-statistic: 6.03 on 7 and 28 DF,  p-value: 0.000237
```

```
par(mfrow = c(1,1))
contour(lm.10.5b.y.SPECx1x2z1z2, ~ x1 + x2, at = data.frame(z1 =  0, z2 =  0), image=TRUE, mai
```




```
par(mfrow = c(2,2))
contour(lm.10.5b.y.SPECx1x2z1z2, ~ x1 + x2, at = data.frame(z1 =  -1, z2 =  -1), image=TRUE, m
contour(lm.10.5b.y.SPECx1x2z1z2, ~ x1 + x2, at = data.frame(z1 =  -1, z2 =   1), image=TRUE, m
contour(lm.10.5b.y.SPECx1x2z1z2, ~ x1 + x2, at = data.frame(z1 =   1, z2 =  -1), image=TRUE, m
contour(lm.10.5b.y.SPECx1x2z1z2, ~ x1 + x2, at = data.frame(z1 =   1, z2 =   1), image=TRUE, m
```


through (z1=−1, z2=−1)
x1
x2
Slice at z1 = −1, z2 = −1
through (z1=−1, z2= 1)
x1
x2
Slice at z1 = −1, z2 = 1
through (z1= 1, z2=−1)
x1
x2
Slice at z1 = 1, z2 = −1
through (z1= 1, z2= 1)
x1
x2
Slice at z1 = 1, z2 = 1

Chapter 11

Experiments with Mixtures

In `rsm()`, mixture designs are not yet provided for.