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
```

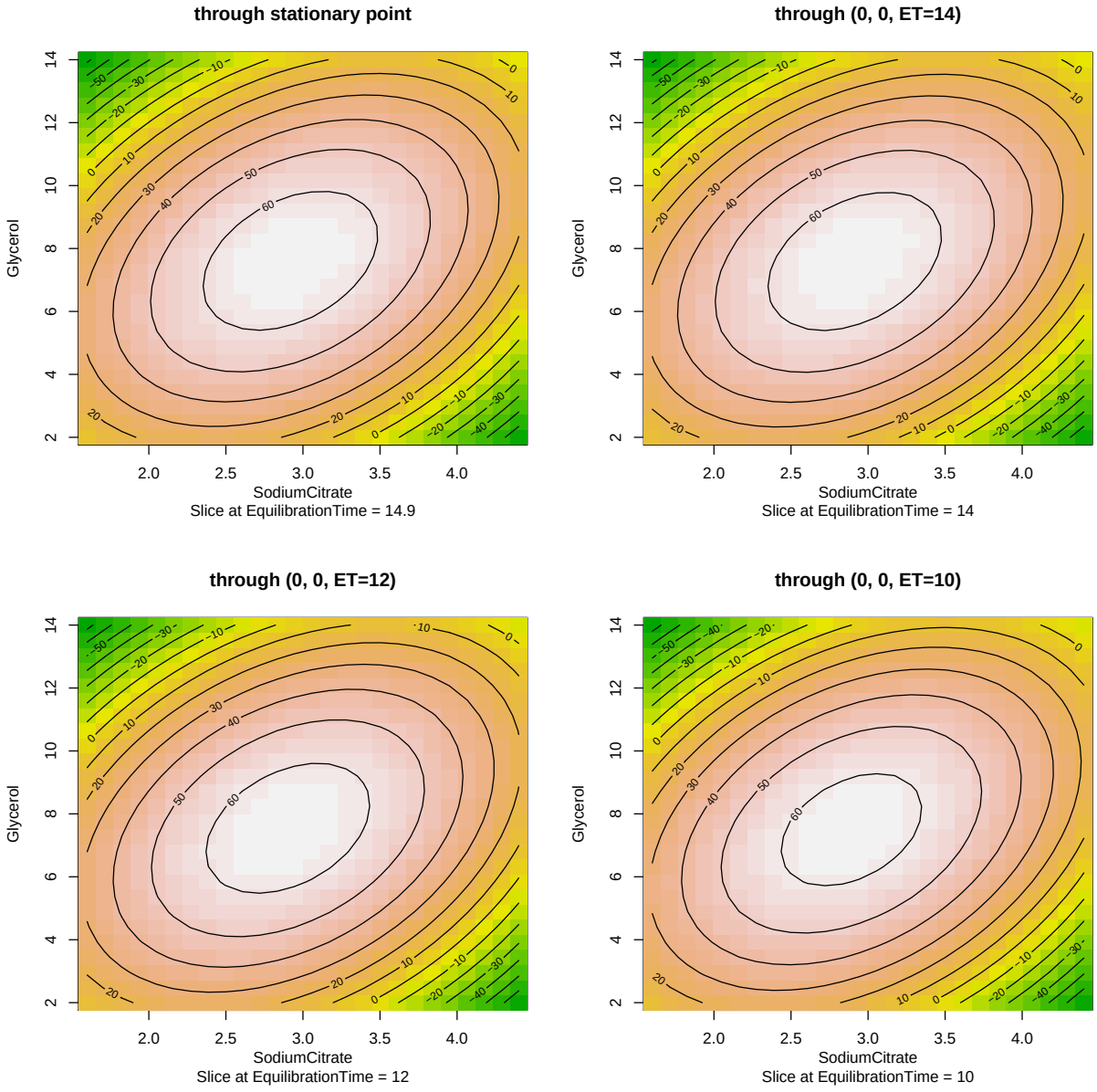
```
## -0.1158 -0.1294 -0.1841
```

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
```

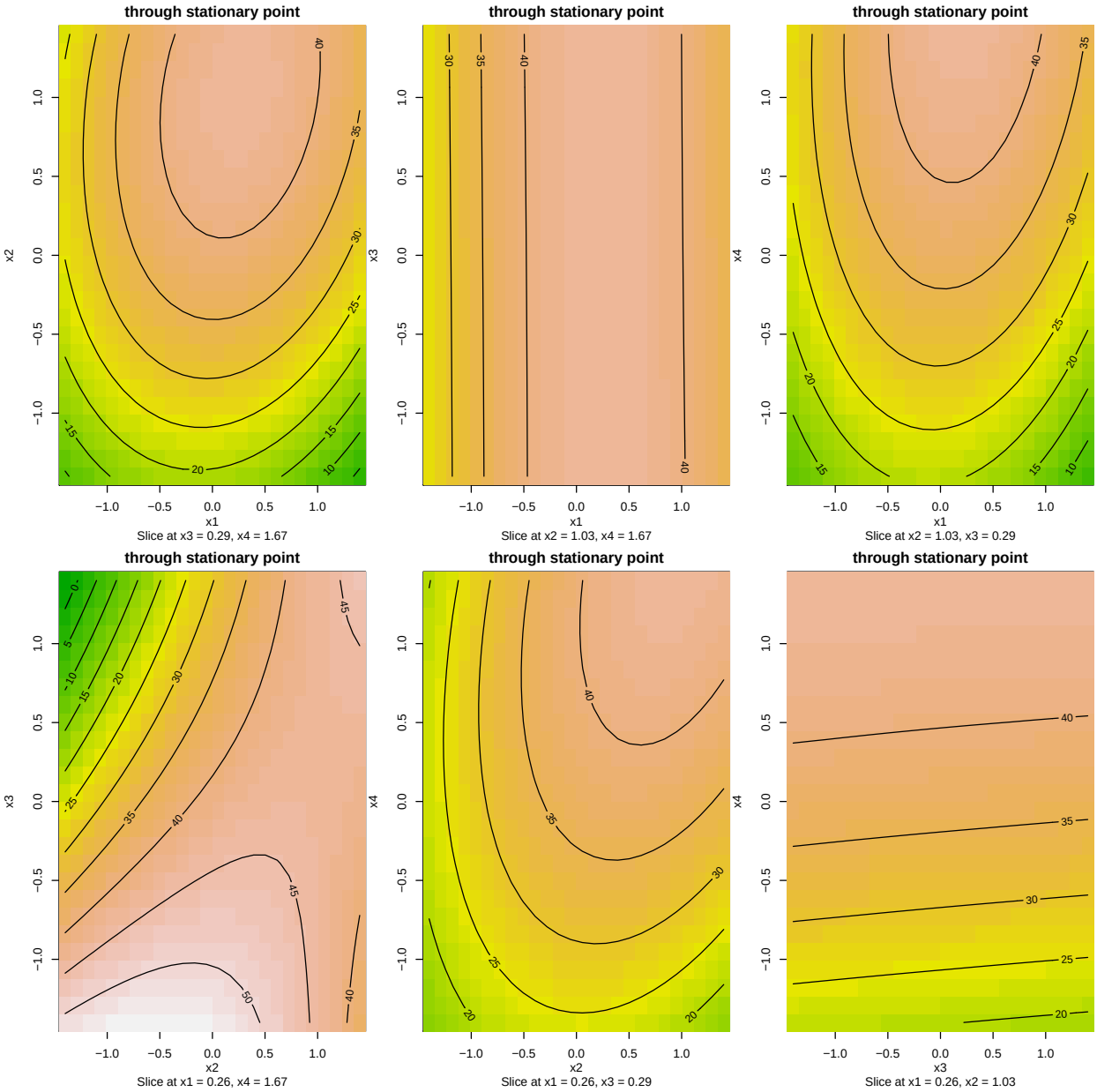
```
canonical(rsm.6.3.y.SOx1x2x3x4)$eigen
```

```
## $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.

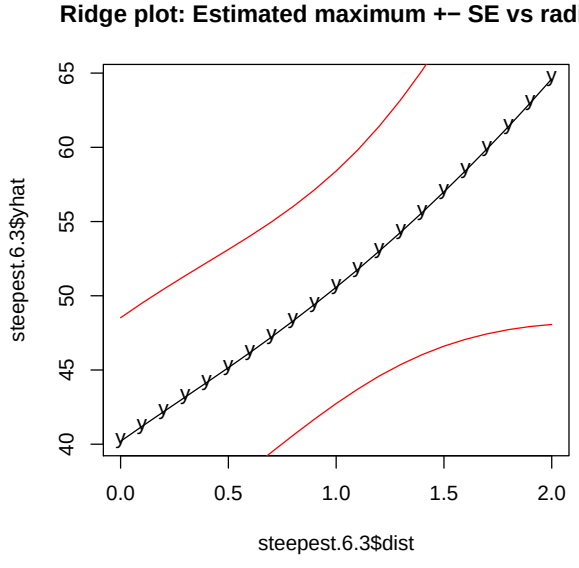
```
par(mfrow = c(2,3), oma = c(0,0,0,0), mar = c(4,4,2,0))
contour(rsm.6.3.y.SOx1x2x3x4, ~ x1 + x2 + x3 + x4, image=TRUE, at = canonical(rsm.6.3.y.SOx1x2
```

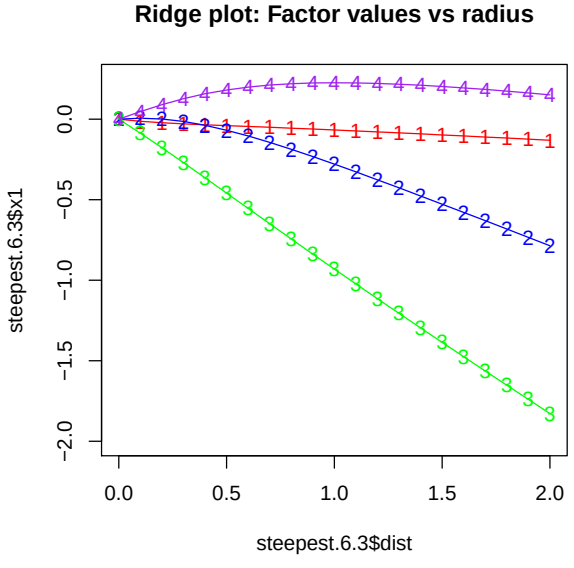


Starting at the stationary point, calculate the predicted increase along the ridge of steepest ascent. Include the standard error of the prediction, as well.

```
# 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)
```

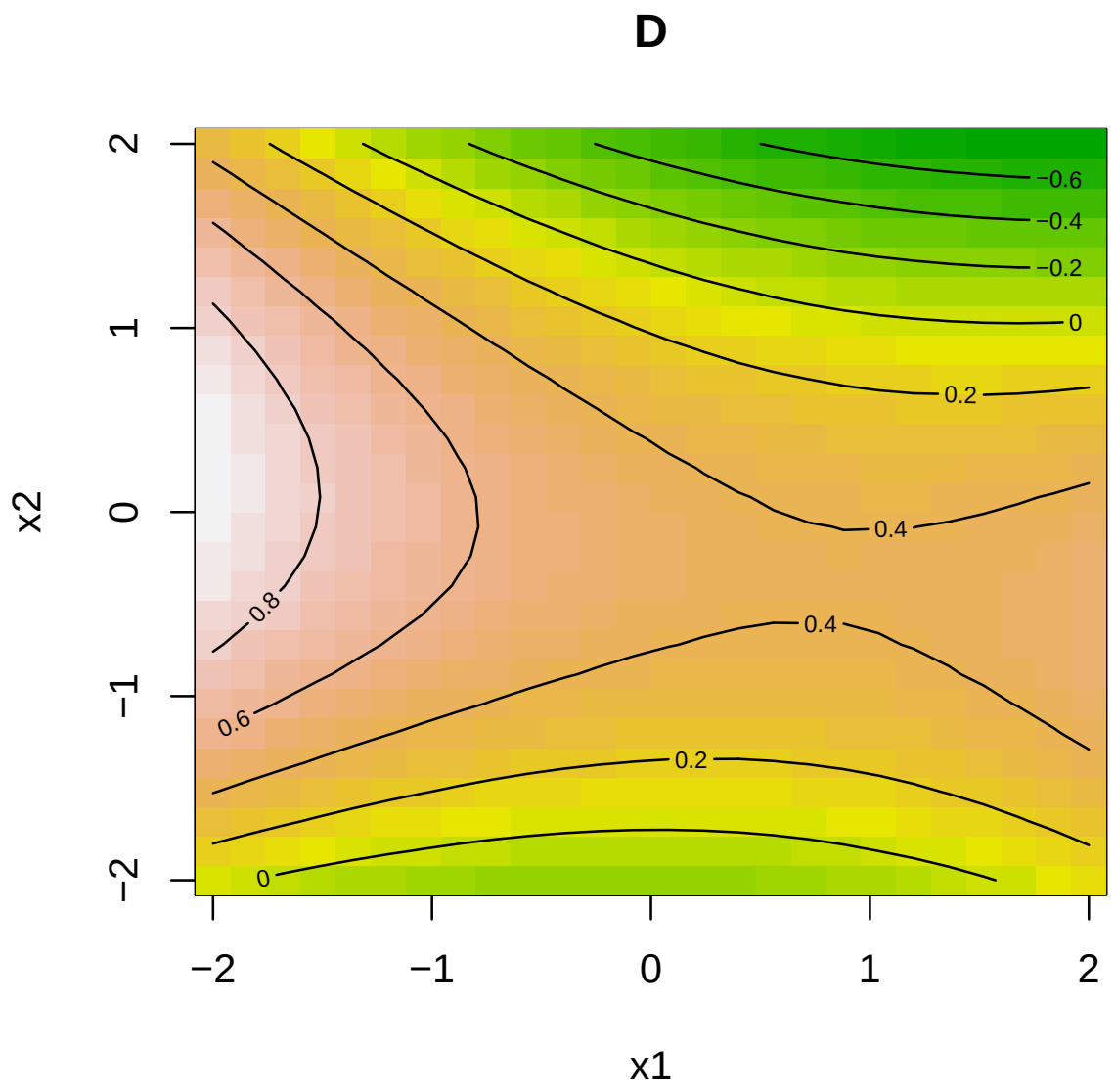
```
##
## Call:
## rsm(formula = D ~ SO(x1, x2), data = df.6.6)
##
```

```
##             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)
```

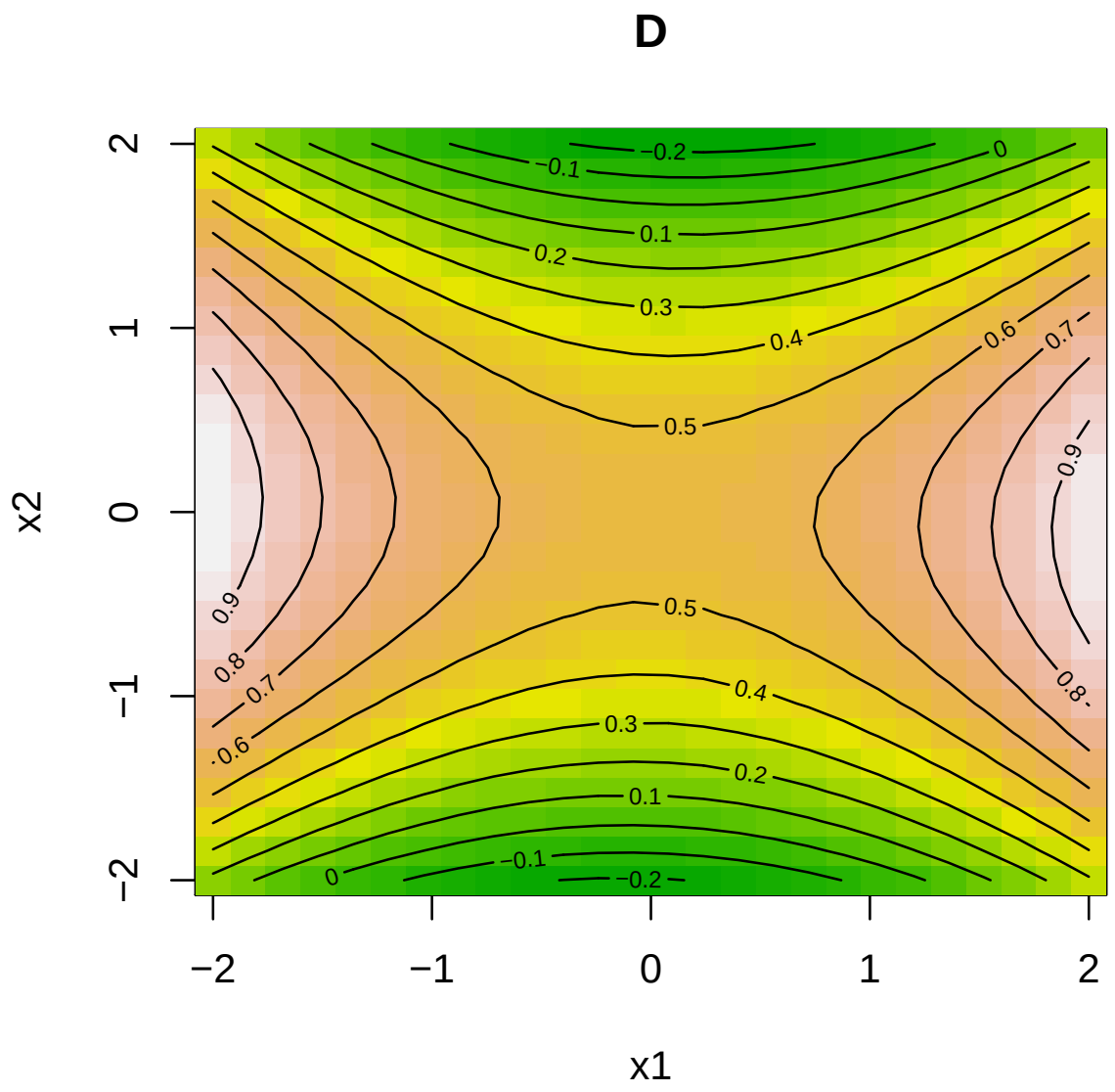
```
##
## Call:
## rsm(formula = D ~ SO(x1, x2), data = df.6.6)
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.54346    0.07896    6.88  0.00023 ***
## x1          -0.00646    0.06243   -0.10  0.92049
## x2          -0.00575    0.06243   -0.09  0.92923
## x1:x2       -0.01748    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.

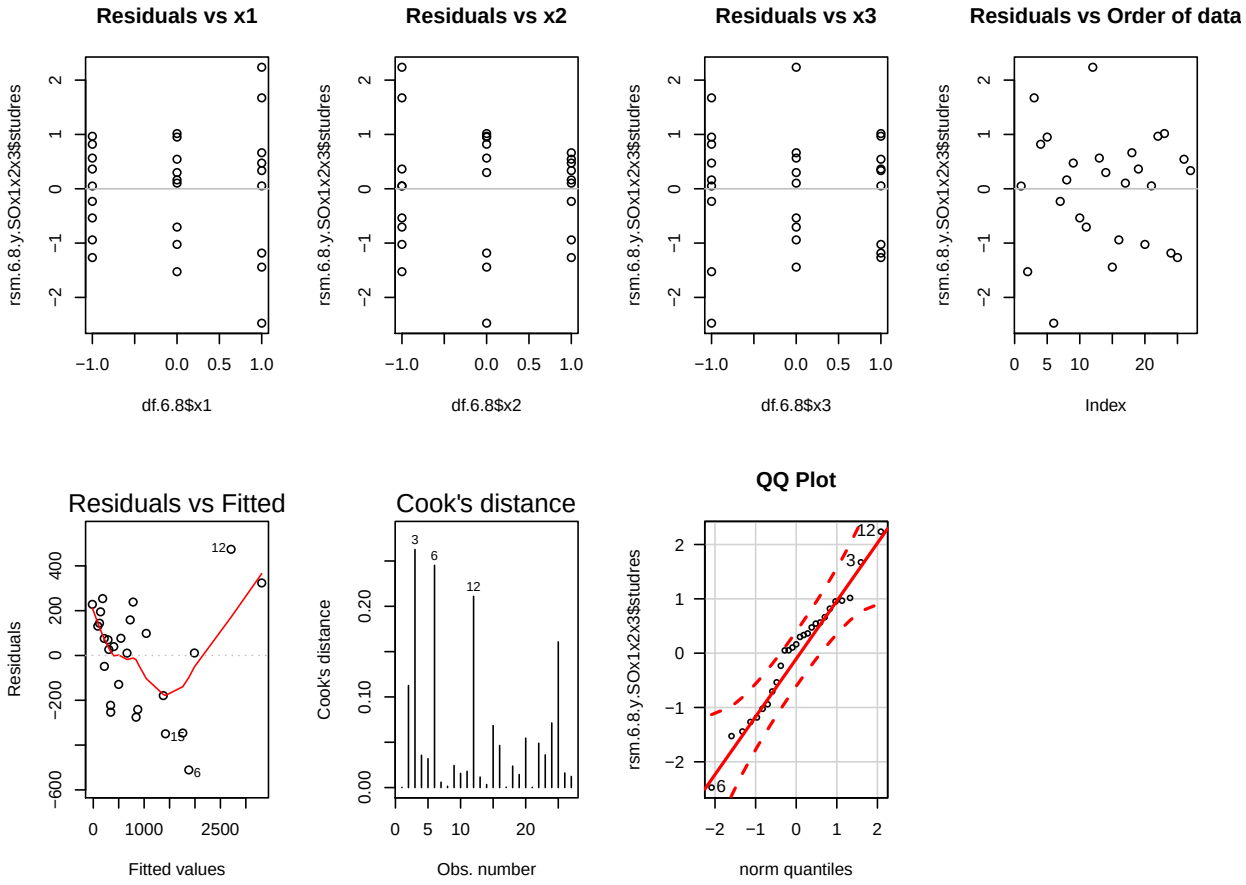
```
# plot diagnistics
par(mfrow=c(2,4))

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)
```

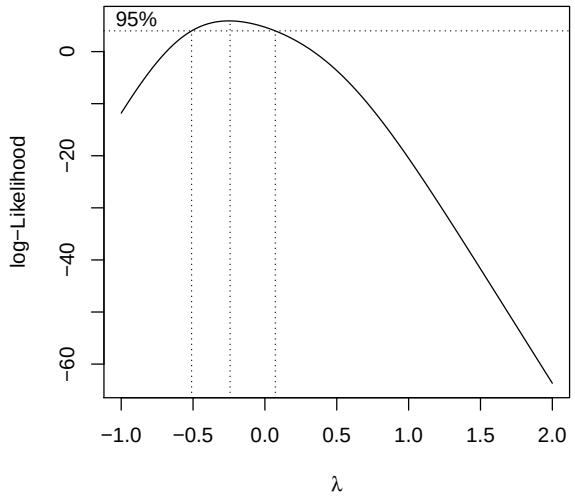
```
##         1         2         3         4         5         6         7
## 0.0002802 0.1126115 0.2626938 0.0356606 0.0318506 0.2451862 0.0059298
```

```
##         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.

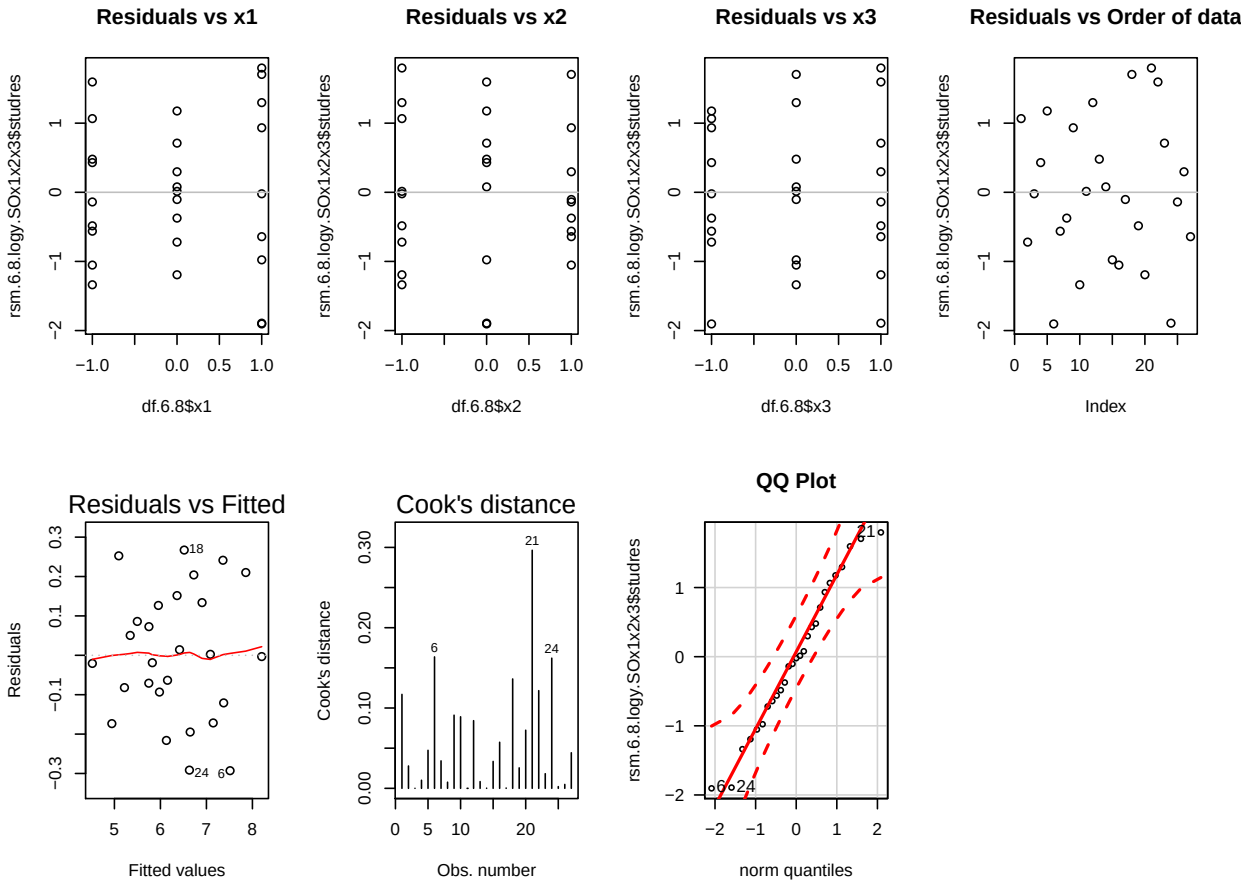
```
# plot diagnistics
par(mfrow=c(2,4))

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
```

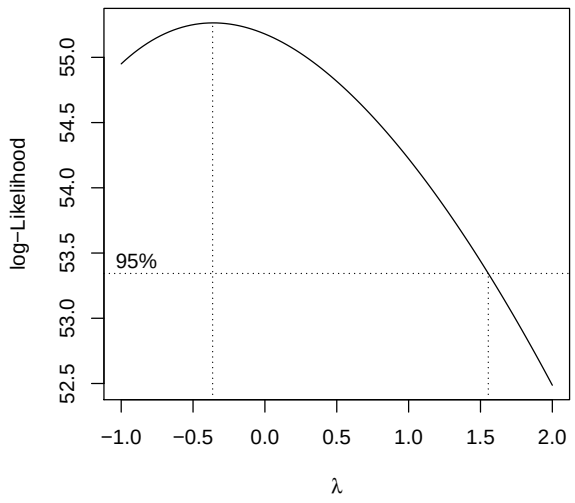
```
cooks.distance(rsm.6.8.logy.SOx1x2x3)
```

```
##         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
52.5
53.0
53.5
54.0
54.5
55.0
95%
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
λ