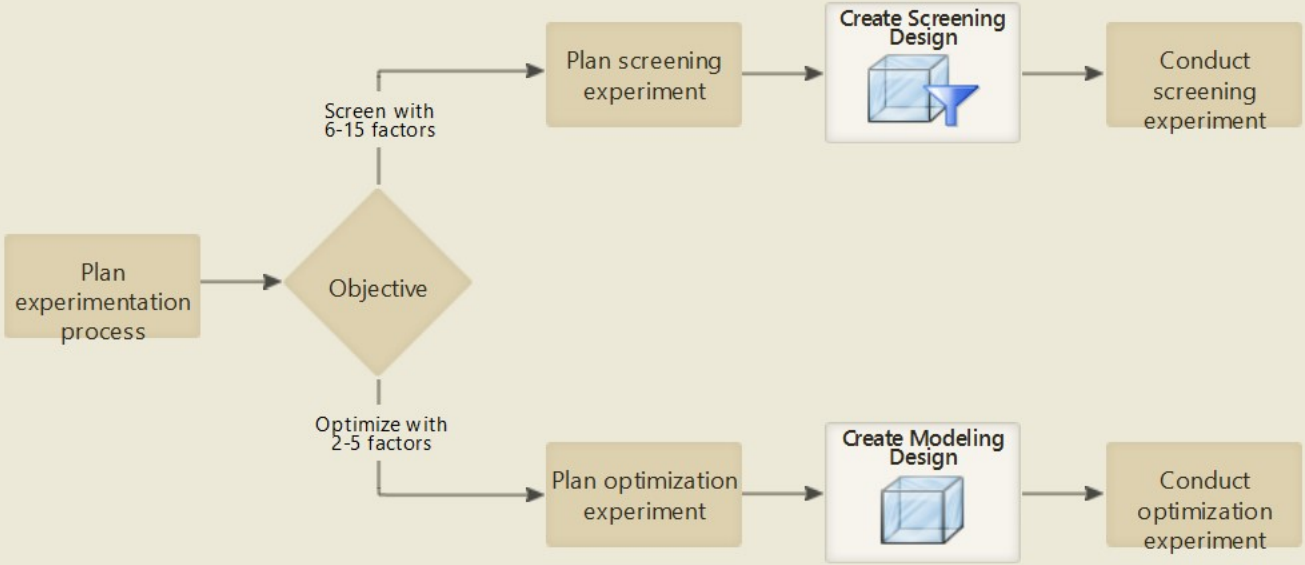

Plan experimentation process
Objective
Screen with 6-15 factors
Plan screening experiment
Create Screening Design
Conduct screening experiment
Optimize with 2-5 factors
Plan optimization experiment
Create Modeling Design
Conduct optimization experiment

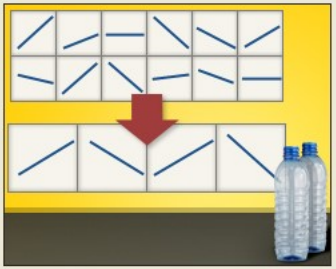
What is your objective?

Screen with 6–15 factors

Examine the main effects of 6–15 factors to identify the critical factors that have the greatest influence on the response. Use a screening experiment when you want to reduce a large number of potential factors.

Example

A plastics manufacturer uses a screening experiment with 12 factors to determine which factors have the greatest influence on the strength of the plastic.



Optimize with 2–5 factors

Construct a model with the 2–5 critical factors that you can use to identify optimal factor settings. Use a modeling design to examine main effects and two-way interactions, and model curvature, if detected.

Example

A food manufacturer creates a modeling design with 4 critical factors to study how the critical factors influence the response.

Back

Plan the Experimentation Process

Before you create a design or collect any data, you should define your objectives and outline a general plan for the experiments you want to conduct. Because resources are limited, it is very important to get the most information from each experiment you perform.

- **Use a sequential experimentation process.**
- **Define the problem you want to address through experimentation.**
- **Define the response (Y) and the goal for Y.**
- **Establish a budget and plan for the process of experimentation.**
- **Communicate the overall plan to stakeholders and obtain approval.**

Plan a Screening Experiment

In the Assistant, you can create a screening design with 6–15 factors. The experimental goal is to identify the critical 2–5 factors that have the greatest influence on the response.

- **Identify the factors.**
- **Define the factor levels.**
- **Determine the appropriate sample size.**
- **Obtain approval from stakeholders.**

Pre-Experiment Checklist for Screening

Before you conduct the experiment, review the following guidelines and complete the appropriate activities.

- **Train individuals involved in the experiment.**
- **Validate measurement system.**
- **Check all design combinations.**
- **Perform trial runs.**

Plan an Optimization Experiment

For the optimization experiment, you need to create a modeling design with the critical 2–5 factors that have the greatest influence on the response. The experimental goal is to determine the settings of the critical factors that will create a desired response, such as a maximum or minimum value, a target value, or a target range. Accomplishing this goal is often a two-step process. Minitab performs a test on the initial modeling design to see if there is curvature in the continuous factors. If curvature is detected, the second step is to add more runs to your design so you can model the curvature and use that model to determine the best settings for the critical factors.

- **Identify the factors.**
- **Define the factor levels.**
- **Determine the sample size.**
- **Obtain approval from stakeholders.**

n and Create Experiments

Pre-Experiment Checklist for Optimization

Before you conduct the experiment, review the following guidelines and complete the appropriate activities.

- Train individuals involved in the experiment.
- Validate measurement system.
- Check all design combinations.
- Perform trial runs.

#packages

#librRIES

```
library(daewr)
```

```
## Registered S3 method overwritten by 'DoE.base':
##   method           from
##   factorize.factor conf.design
```

```
library(DoE.base)
```

```
## Loading required package: grid
```

```
## Loading required package: conf.design
```

```
##
## Attaching package: 'DoE.base'
```

```
## The following objects are masked from 'package:stats':
##
##     aov, lm
```

```
## The following object is masked from 'package:graphics':
##
##     plot.design
```

```
## The following object is masked from 'package:base':
##
##     lengths
```

```
library(FrF2)
library(rsm)
```

#SCREENING PARAMETERS

#FULL vs Factorial Factorial design

```
ff23 <- FrF2( 8, 3, randomize = FALSE)
```

```
## creating full factorial with 8 runs ...
```

```
ff23
```

```
##    A  B  C
## 1 -1 -1 -1
## 2  1 -1 -1
## 3 -1  1 -1
## 4  1  1 -1
## 5 -1 -1  1
## 6  1 -1  1
## 7 -1  1  1
## 8  1  1  1
## class=design, type= full factorial
```

```
ff23f <- FrF2( 4, 3, randomize = FALSE)
ff23f
```

```
##    A  B  C
## 1 -1 -1  1
## 2  1 -1 -1
## 3 -1  1 -1
## 4  1  1  1
## class=design, type= FrF2
```

```
y <- runif(4, 0, 1)
aliases( lm( y~ (.)^4, data = ff23f))
```

```
##
##  A = B:C
##  B = A:C
##  C = A:B
```

```
design <- FrF2( 16, 5, generators = "ABCD", randomize = FALSE)
design
```

```
##     A  B  C  D  E
## 1  -1 -1 -1 -1  1
## 2   1 -1 -1 -1 -1
## 3  -1  1 -1 -1 -1
## 4   1  1 -1 -1  1
## 5  -1 -1  1 -1 -1
## 6   1 -1  1 -1  1
## 7  -1  1  1 -1  1
## 8   1  1  1 -1 -1
## 9  -1 -1 -1  1 -1
## 10  1 -1 -1  1  1
## 11 -1  1 -1  1  1
## 12  1  1 -1  1 -1
## 13 -1 -1  1  1  1
## 14  1 -1  1  1 -1
## 15 -1  1  1  1 -1
## 16  1  1  1  1  1
## class=design, type= FrF2.generators
```

```
y <- runif(16, 0, 1)
aliases( lm( y~ (.)^4, data = design))
```

```
##
##  A = B:C:D:E
##  B = A:C:D:E
##  C = A:B:D:E
##  D = A:B:C:E
##  E = A:B:C:D
##  A:B = C:D:E
##  A:C = B:D:E
##  A:D = B:C:E
##  A:E = B:C:D
##  B:C = A:D:E
##  B:D = A:C:E
##  B:E = A:C:D
##  C:D = A:B:E
##  C:E = A:B:D
##  D:E = A:B:C
```

#Plackett-Burman designs it can be created easily using the FrF2 package. The example below illustrates the use of the pb function in that package to create the design with 11 factors using 12 runs.

```
library(FrF2)
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
```

#Exercise full factorial

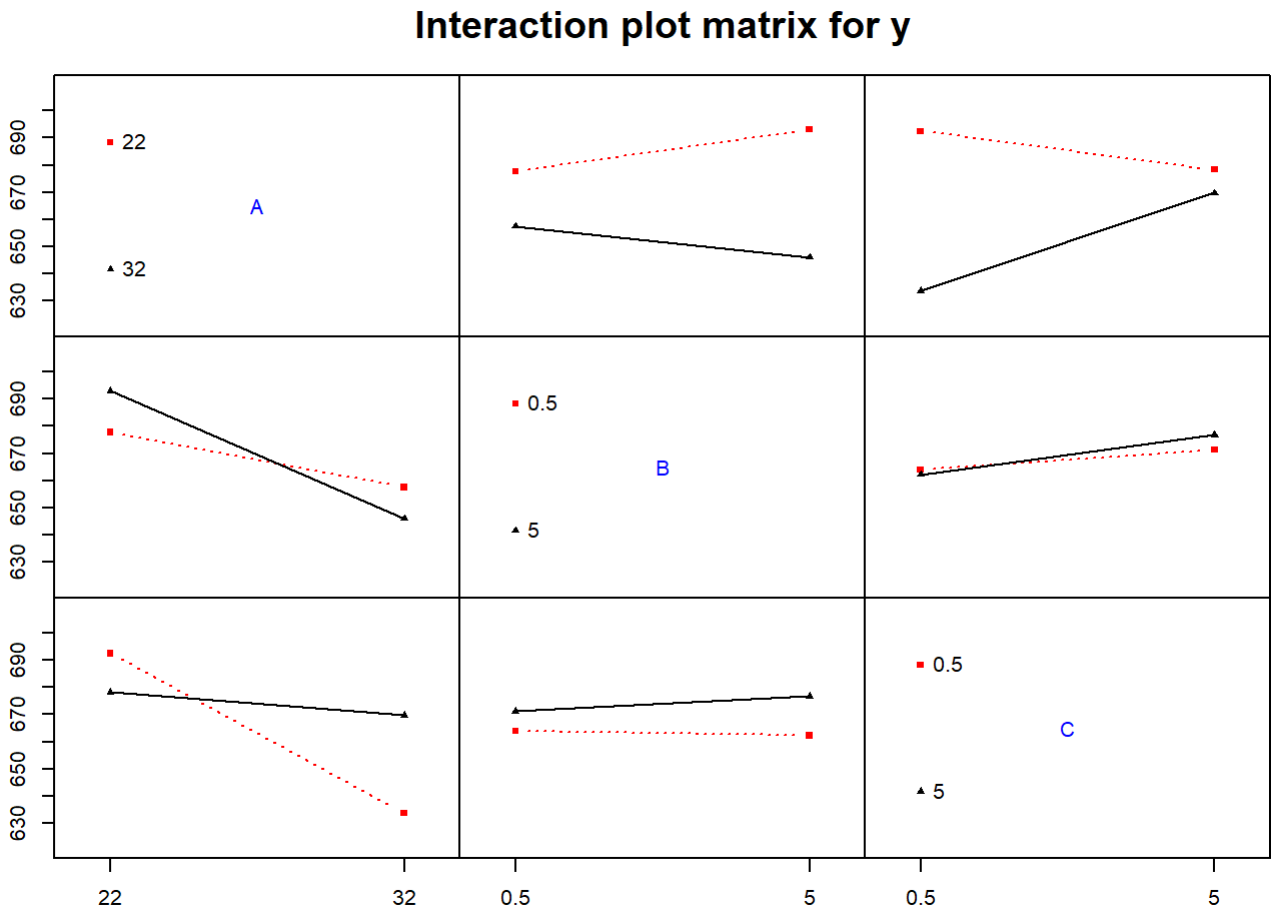
```
volt
```

```
##     A   B   C   y
## 1  22 0.5 0.5 705
## 2  32 0.5 0.5 620
## 3  22   5 0.5 700
## 4  32   5 0.5 629
## 5  22 0.5   5 672
## 6  32 0.5   5 668
## 7  22   5   5 715
## 8  32   5   5 647
## 9  22 0.5 0.5 680
## 10 32 0.5 0.5 651
## 11 22   5 0.5 685
## 12 32   5 0.5 635
## 13 22 0.5   5 654
## 14 32 0.5   5 691
## 15 22   5   5 672
## 16 32   5   5 673
```

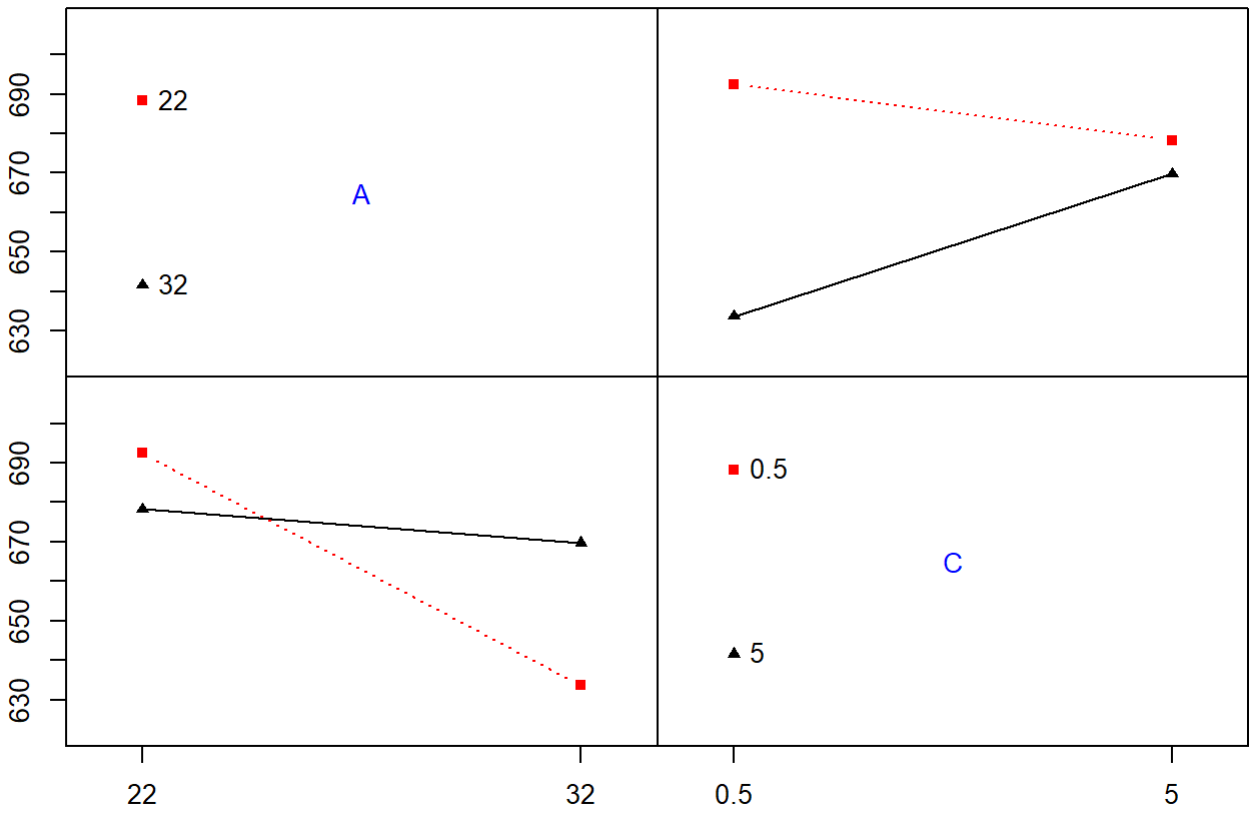
```
modv <- lm( y ~ A*B*C, data=volt, contrast=list(A=contr.FrF2, B=contr.FrF2, C=contr.FrF2))
summary(modv)
```

```
##
## Call:
## lm.default(formula = y ~ A * B * C, data = volt, contrasts = list(A = contr.FrF2,
##     B = contr.FrF2, C = contr.FrF2))
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -21.50 -11.75   0.00  11.75  21.50
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 668.5625     4.5178 147.985 4.86e-15 ***
## A1          -16.8125     4.5178  -3.721  0.00586 **
## B1            0.9375     4.5178   0.208  0.84079
## C1            5.4375     4.5178   1.204  0.26315
## A1:B1        -6.6875     4.5178  -1.480  0.17707
## A1:C1        12.5625     4.5178   2.781  0.02390 *
## B1:C1         1.8125     4.5178   0.401  0.69878
## A1:B1:C1     -5.8125     4.5178  -1.287  0.23422
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 18.07 on 8 degrees of freedom
## Multiple R-squared:  0.772,  Adjusted R-squared:  0.5724
## F-statistic: 3.869 on 7 and 8 DF,  p-value: 0.0385
```

```
par( mfrow = c(2,2) )
IAPlot(modv)
```




```
IAPlot(modv, select = c(1,3))
```



#Exercise full factorial

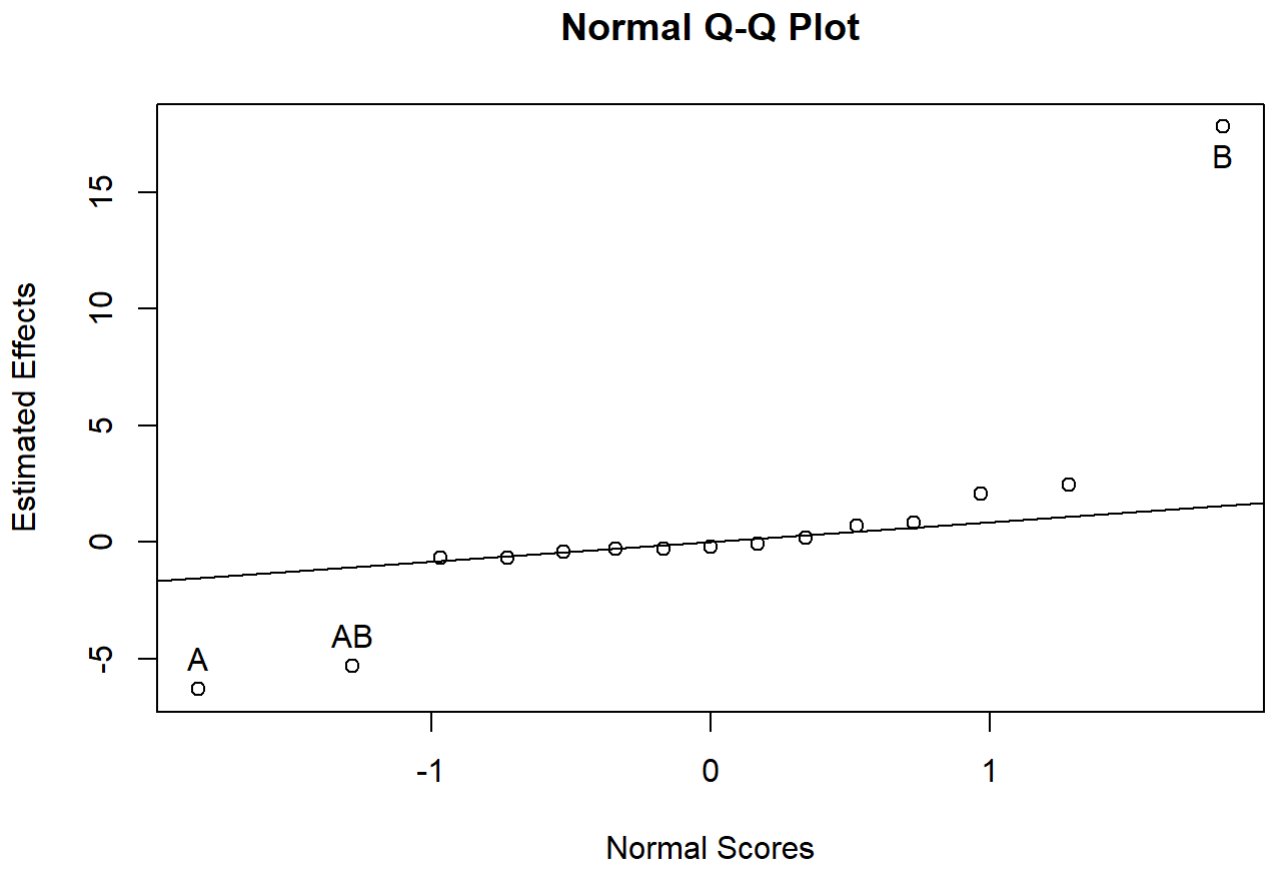
```
chem
```

```
##     A  B  C  D  y
## 1  -1 -1 -1 -1 45
## 2   1 -1 -1 -1 41
## 3  -1  1 -1 -1 90
## 4   1  1 -1 -1 67
## 5  -1 -1  1 -1 50
## 6   1 -1  1 -1 39
## 7  -1  1  1 -1 95
## 8   1  1  1 -1 66
## 9  -1 -1 -1  1 47
## 10  1 -1 -1  1 43
## 11 -1  1 -1  1 95
## 12  1  1 -1  1 69
## 13 -1 -1  1  1 40
## 14  1 -1  1  1 51
## 15 -1  1  1  1 87
## 16  1  1  1  1 72
```

```
modf <- lm( y ~ A*B*C*D, data = chem)
 summary(modf)
```

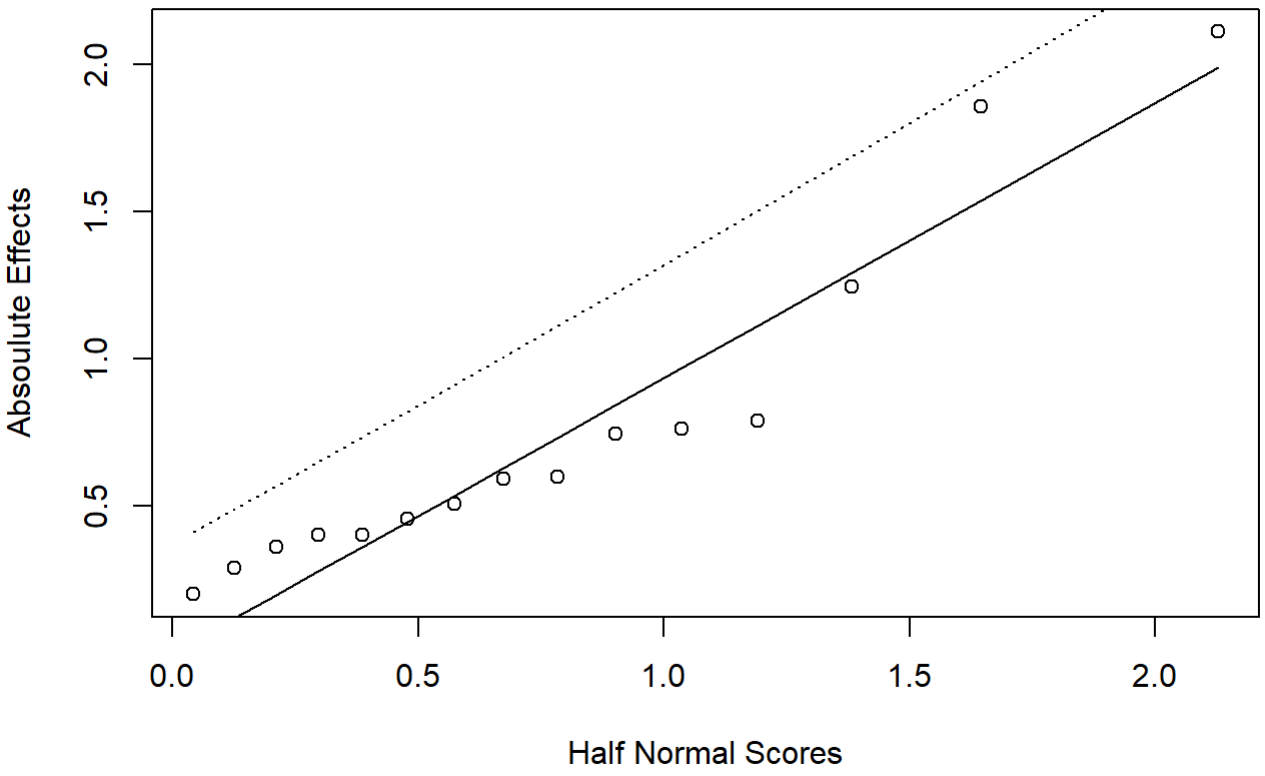
```
##
## Call:
## lm.default(formula = y ~ A * B * C * D, data = chem)
##
## Residuals:
## ALL 16 residuals are 0: no residual degrees of freedom!
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  62.3125        NaN     NaN      NaN
## A            -6.3125        NaN     NaN      NaN
## B            17.8125        NaN     NaN      NaN
## C             0.1875        NaN     NaN      NaN
## D             0.6875        NaN     NaN      NaN
## A:B          -5.3125        NaN     NaN      NaN
## A:C           0.8125        NaN     NaN      NaN
## B:C          -0.3125        NaN     NaN      NaN
## A:D           2.0625        NaN     NaN      NaN
## B:D          -0.0625        NaN     NaN      NaN
## C:D          -0.6875        NaN     NaN      NaN
## A:B:C        -0.1875        NaN     NaN      NaN
## A:B:D        -0.6875        NaN     NaN      NaN
## A:C:D         2.4375        NaN     NaN      NaN
## B:C:D        -0.4375        NaN     NaN      NaN
## A:B:C:D      -0.3125        NaN     NaN      NaN
##
## Residual standard error: NaN on 0 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:    NaN
## F-statistic:   NaN on 15 and 0 DF,  p-value: NA
```

```
fullnormal(coef(modf)[-1],alpha=.025)
```

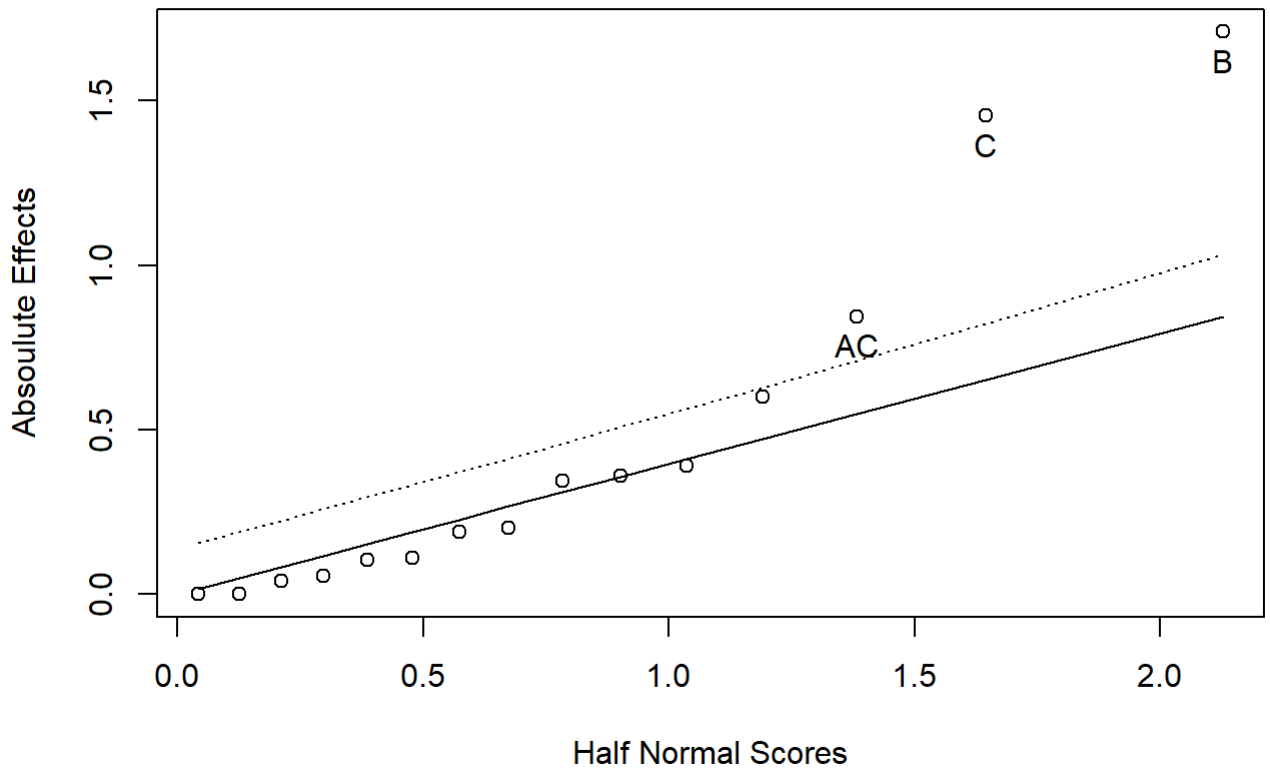


#Exercise full factorial

```
data(BoxM)
Gaptest(BoxM)
```


Absoulute Effects
0.5
1.0
1.5
2.0
0.0
0.5
1.0
1.5
2.0
Half Normal Scores

```
## Effect Report
##
## Label     Half Effect    Sig(.05)
## A          -0.400        no
## B          -2.110        no
## C           1.855        no
## D           0.505        no
## AB          0.455        no
## AC         -1.245        no
## AD         -0.290        no
## BC         -0.400        no
## BD         -0.590        no
## CD          0.745        no
## ABC         0.600        no
## ABD         0.360        no
## ACD         0.200        no
## BCD        -0.790        no
## ABCD        0.760        no
##
## Lawson, Grimshaw & Burt Rn Statistic =  1
## 95th percentile of Rn =  1.201
## Initial Outlier Report
## Standardized-Gap =  3.353227 Significant at 50th percentile
## Final Outlier Report
## Standardized-Gap =  13.18936 Significant at 99th percentile
##
##     Corrrected Data Report
## Response  Corrected Response   Detect Outlier
##    47.46         47.46             no
##    49.62         49.62             no
##    43.13         43.13             no
##    46.31         46.31             no
##    51.47         51.47             no
##    48.49         48.49             no
##    49.34         49.34             no
##    46.10         46.10             no
##    46.76         46.76             no
##    48.56         48.56             no
##    44.83         44.83             no
##    44.45         44.45             no
##    59.15         52.75             yes
##    51.33         51.33             no
##    47.02         47.02             no
##    47.90         47.90             no
```




```
## Effect Report
##
## Label     Half Effect    Sig(.05)
## A        -4.514306e-15         no
## B        -1.710000e+00        yes
## C         1.455000e+00        yes
## D         1.050000e-01         no
## AB        5.500000e-02         no
## AC       -8.450000e-01        yes
## AD        1.100000e-01         no
## BC        2.170070e-15         no
## BD       -1.900000e-01         no
## CD        3.450000e-01         no
## ABC       2.000000e-01         no
## ABD      -4.000000e-02         no
## ACD       6.000000e-01         no
## BCD      -3.900000e-01         no
## ABCD      3.600000e-01         no
##
## Lawson, Grimshaw & Burt Rn Statistic =  1.626089
## 95th percentile of Rn =  1.201
```

#Exercise fractional factorial

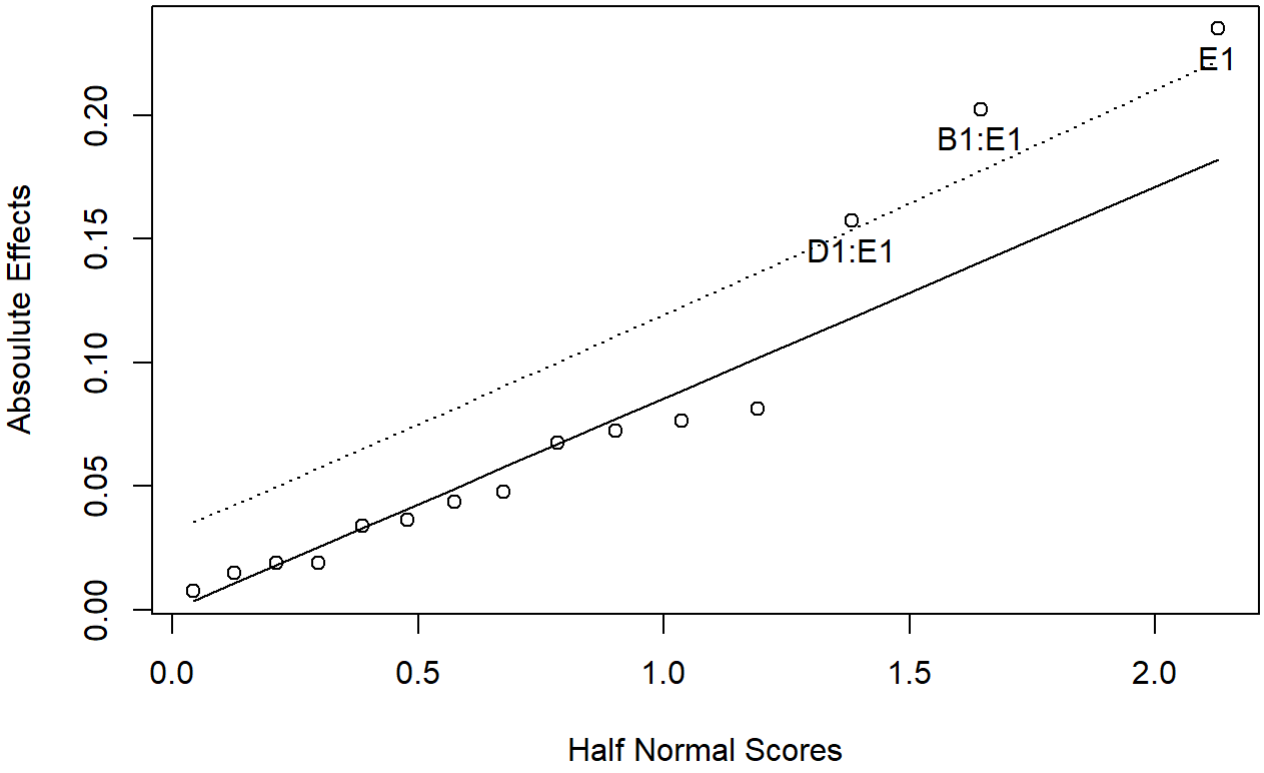
```
soup <- FrF2(16, 5, generators = "ABCD", factor.names =list(Ports=c(1,3), Temp=c("Cool","Ambien
t"), MixTime=c(60,80),BatchWt=c(1500,2000), delay=c(7,1)), randomize = FALSE)
y <- c(1.13, 1.25, .97, 1.70, 1.47, 1.28, 1.18, .98, .78, 1.36, 1.85, .62, 1.09, 1.10, .76, 2.10
)
soup <- add.response( soup , y )
mod1 <- lm( y ~ (.)^2, data = soup)
mod1
```

```
##
## Call:
## lm.default(formula = y ~ (.)^2, data = soup)
##
## Coefficients:
##       (Intercept)             Ports1              Temp1           MixTime1
##           1.22625            0.07250            0.04375            0.01875
##          BatchWt1             delay1       Ports1:Temp1    Ports1:MixTime1
##          -0.01875            0.23500            0.00750            0.04750
##   Ports1:BatchWt1      Ports1:delay1     Temp1:MixTime1     Temp1:BatchWt1
##           0.01500            0.07625           -0.03375            0.08125
##      Temp1:delay1  MixTime1:BatchWt1    MixTime1:delay1    BatchWt1:delay1
##           0.20250            0.03625           -0.06750            0.15750
```

```
summary(mod1)
```

```
##
## Call:
## lm.default(formula = y ~ (.)^2, data = soup)
##
## Residuals:
## ALL 16 residuals are 0: no residual degrees of freedom!
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)        1.22625        NaN     NaN      NaN
## Ports1             0.07250        NaN     NaN      NaN
## Temp1              0.04375        NaN     NaN      NaN
## MixTime1           0.01875        NaN     NaN      NaN
## BatchWt1          -0.01875        NaN     NaN      NaN
## delay1             0.23500        NaN     NaN      NaN
## Ports1:Temp1       0.00750        NaN     NaN      NaN
## Ports1:MixTime1    0.04750        NaN     NaN      NaN
## Ports1:BatchWt1    0.01500        NaN     NaN      NaN
## Ports1:delay1      0.07625        NaN     NaN      NaN
## Temp1:MixTime1    -0.03375        NaN     NaN      NaN
## Temp1:BatchWt1     0.08125        NaN     NaN      NaN
## Temp1:delay1       0.20250        NaN     NaN      NaN
## MixTime1:BatchWt1  0.03625        NaN     NaN      NaN
## MixTime1:delay1   -0.06750        NaN     NaN      NaN
## BatchWt1:delay1    0.15750        NaN     NaN      NaN
##
## Residual standard error: NaN on 0 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:    NaN
## F-statistic:   NaN on 15 and 0 DF,  p-value: NA
```

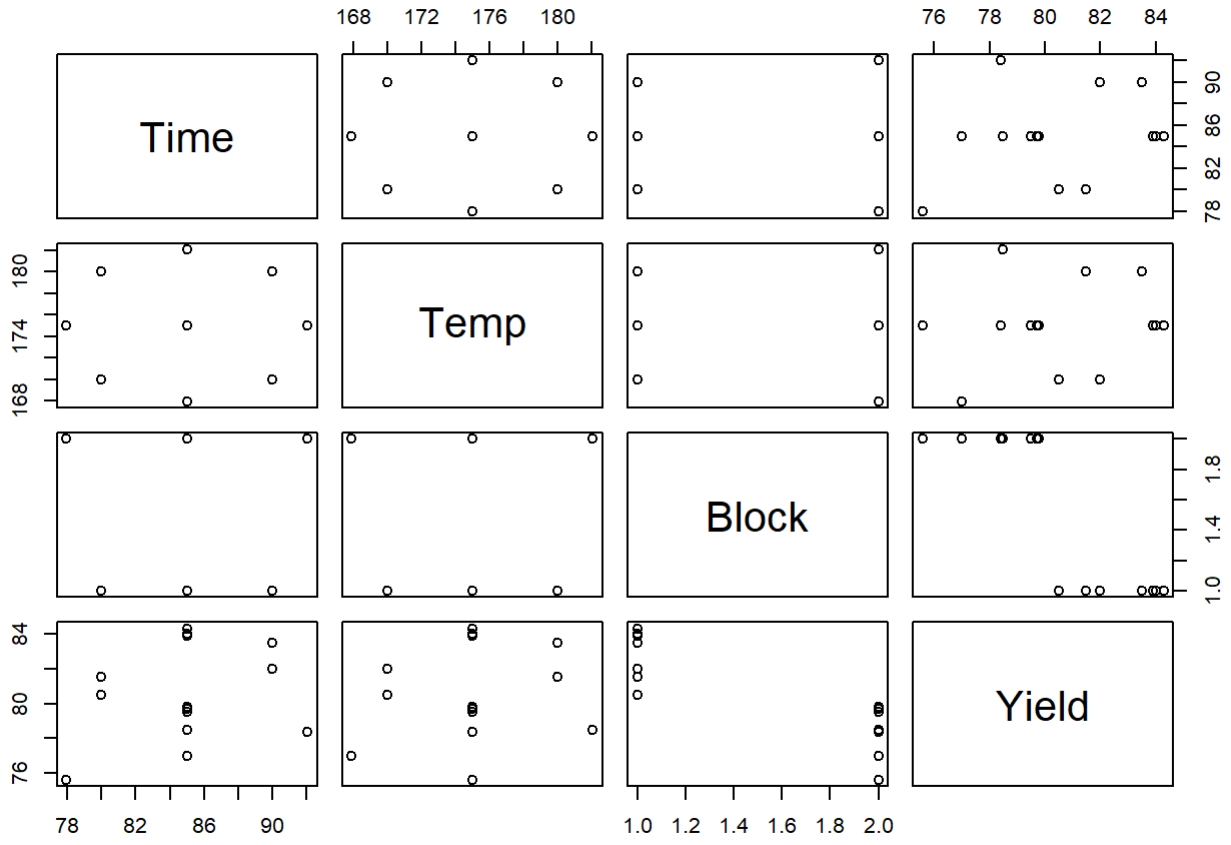
```
soupc<-FrF2(16,5,generators="ABCD",randomize=FALSE)
soupc<-add.response(soupc, y)



soupc<-FrF2(16,5,generators="ABCD",randomize=FALSE)
soupc<-add.response(soupc, y)

modc<-lm(y~(.)^2, data=soupc)
LGB(coef(modc)[-1], rpt = FALSE)
```




```
#Exercise central composite designs 2 factors
#https://www.youtube.com/watch?v=5Zb-3gZlL1E
# Lez74 http://www.lithoguru.com/scientist/statistics/course.html


#-------------------------------------#
#--- Response Surface Modeling in R ---#
#-------------------------------------#


#First, install and load the "rsm" package


# install.packages("rsm")
library(rsm)


# Example generating a Box-Behnken design with three factors and two center points (no)
bbd(3, n0 = 2, coding = list(x1 ~ (Force - 20)/3, x2 ~ (Rate - 50)/10, x3 ~ Polish - 4))
```

```
##    run.order std.order Force Rate Polish
## 1          1          9    20   40      3
## 2          2          4    23   60      4
## 3          3          8    23   50      5
## 4          4          3    17   60      4
## 5          5         13    20   50      4
## 6          6          5    17   50      3
## 7          7         14    20   50      4
## 8          8          2    23   40      4
## 9          9         12    20   60      5
## 10        10         10    20   60      3
## 11        11          1    17   40      4
## 12        12          7    17   50      5
## 13        13          6    23   50      3
## 14        14         11    20   40      5
##
## Data are stored in coded form using these coding formulas ...
## x1 ~ (Force - 20)/3
## x2 ~ (Rate - 50)/10
## x3 ~ Polish - 4
```

```
# Example data set
data = ChemReact
plot(data)
```



```
# The data set was collected in two blocks.
# Block1 is a 2-level, two-factor factorial design with three repeated center points.
# Block 2 is the Central Composite Design (circomscribed) with 3 center points.
# The variables are Time = 85 +/- 5 and Temp = 175 +/- 5,
# Thus, the coded variables are  x1 = (Time-85)/5 and x2 = (Temp-175)/5
CR <- coded.data(ChemReact, x1 ~ (Time - 85)/5, x2 ~ (Temp - 175)/5)
CR[1:7,]
```

```
##   Time Temp Block Yield
## 1   80  170    B1  80.5
## 2   80  180    B1  81.5
## 3   90  170    B1  82.0
## 4   90  180    B1  83.5
## 5   85  175    B1  83.9
## 6   85  175    B1  84.3
## 7   85  175    B1  84.0
##
## Data are stored in coded form using these coding formulas ...
## x1 ~ (Time - 85)/5
## x2 ~ (Temp - 175)/5
```

```
# Note: If the data are already coded, use as.coded.data() to convert to the proper coded data o
bject


# Let's work as though the first block (full factorial) has been finished,
# and we'll fit a linear model, first order (FO), to it (Yield is the response)
CR.rsm1 <- rsm(Yield ~ FO(x1, x2), data = CR, subset = (Block == "B1"))
summary(CR.rsm1)
```

```
##
## Call:
## rsm(formula = Yield ~ FO(x1, x2), data = CR, subset = (Block ==
##     "B1"))
##
##             Estimate Std. Error  t value  Pr(>|t|)
## (Intercept) 82.81429    0.54719 151.3456 1.143e-08 ***
## x1           0.87500    0.72386   1.2088    0.2933
## x2           0.62500    0.72386   0.8634    0.4366
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.3555, Adjusted R-squared:  0.0333
## F-statistic: 1.103 on 2 and 4 DF,  p-value: 0.4153
##
## Analysis of Variance Table
##
## Response: Yield
##             Df Sum Sq Mean Sq F value  Pr(>F)
## FO(x1, x2)   2 4.6250  2.3125  1.1033 0.41534
## Residuals    4 8.3836  2.0959
## Lack of fit  2 8.2969  4.1485 95.7335 0.01034
## Pure error   2 0.0867  0.0433
##
## Direction of steepest ascent (at radius 1):
##        x1        x2
## 0.8137335 0.5812382
##
## Corresponding increment in original units:
##     Time     Temp
## 4.068667 2.906191
```

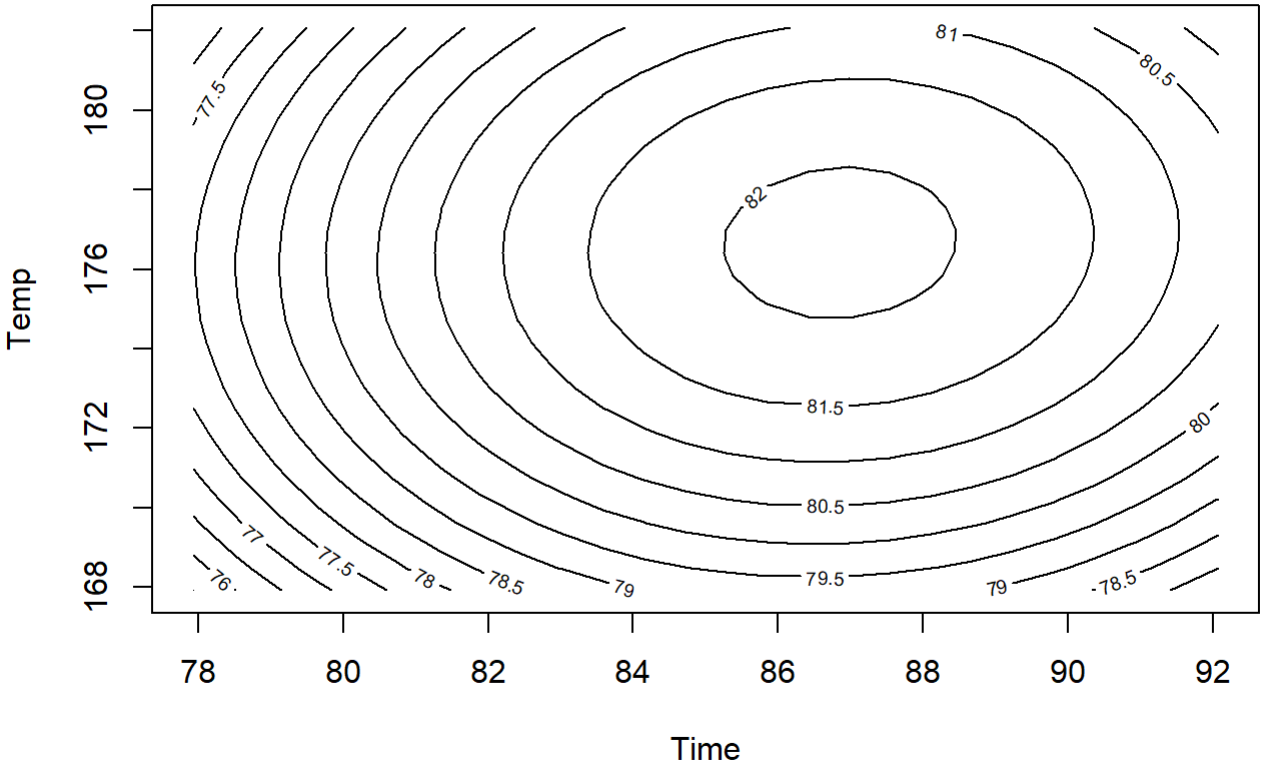
```
#The fit is not very good.  Let's include the interaction term (TWI) and update the model, or st
art over with a new model (these two lines do the same thing)
CR.rsm1.5 <- update(CR.rsm1, . ~ . + TWI(x1, x2))
CR.rsm1.5 <- rsm(Yield ~ FO(x1, x2)+TWI(x1, x2), data = CR, subset = (Block == "B1"))
summary(CR.rsm1.5)
```

```
##
## Call:
## rsm(formula = Yield ~ FO(x1, x2) + TWI(x1, x2), data = CR, subset = (Block ==
##     "B1"))
##
##             Estimate Std. Error  t value  Pr(>|t|)
## (Intercept) 82.81429    0.62948 131.5604 9.683e-07 ***
## x1           0.87500    0.83272   1.0508    0.3705
## x2           0.62500    0.83272   0.7506    0.5074
## x1:x2        0.12500    0.83272   0.1501    0.8902
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.3603, Adjusted R-squared:  -0.2793
## F-statistic: 0.5633 on 3 and 3 DF,  p-value: 0.6755
##
## Analysis of Variance Table
##
## Response: Yield
##             Df Sum Sq Mean Sq  F value   Pr(>F)
## FO(x1, x2)   2 4.6250  2.3125   0.8337 0.515302
## TWI(x1, x2)  1 0.0625  0.0625   0.0225 0.890202
## Residuals    3 8.3211  2.7737
## Lack of fit  1 8.2344  8.2344 190.0247 0.005221
## Pure error   2 0.0867  0.0433
##
## Stationary point of response surface:
## x1 x2
## -5 -7
##
## Stationary point in original units:
## Time Temp
##   60  140
##
## Eigenanalysis:
## eigen() decomposition
## $values
## [1]  0.0625 -0.0625
##
## $vectors
##         [,1]       [,2]
## x1 0.7071068 -0.7071068
## x2 0.7071068  0.7071068
```

```
#This is no better!  The reason is the strong quadratic response, with the peak near the center.


# Now let's assume the second block has been collected.  We use the SO (second order) function,
 which includes FO and TWI
CR.rsm2 <- rsm(Yield ~ Block + SO(x1, x2), data = CR)
summary(CR.rsm2)
```

```
##
## Call:
## rsm(formula = Yield ~ Block + SO(x1, x2), data = CR)
##
##              Estimate Std. Error  t value  Pr(>|t|)
## (Intercept) 84.095427   0.079631 1056.067 < 2.2e-16 ***
## BlockB2     -4.457530   0.087226  -51.103 2.877e-10 ***
## x1           0.932541   0.057699   16.162 8.444e-07 ***
## x2           0.577712   0.057699   10.012 2.122e-05 ***
## x1:x2        0.125000   0.081592    1.532    0.1694
## x1^2        -1.308555   0.060064  -21.786 1.083e-07 ***
## x2^2        -0.933442   0.060064  -15.541 1.104e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.9981, Adjusted R-squared:  0.9964
## F-statistic: 607.2 on 6 and 7 DF,  p-value: 3.811e-09
##
## Analysis of Variance Table
##
## Response: Yield
##             Df Sum Sq Mean Sq   F value    Pr(>F)
## Block        1 69.531  69.531 2611.0950 2.879e-10
## FO(x1, x2)   2  9.626   4.813  180.7341 9.450e-07
## TWI(x1, x2)  1  0.063   0.063    2.3470    0.1694
## PQ(x1, x2)   2 17.791   8.896  334.0539 1.135e-07
## Residuals    7  0.186   0.027
## Lack of fit  3  0.053   0.018    0.5307    0.6851
## Pure error   4  0.133   0.033
##
## Stationary point of response surface:
##        x1        x2
## 0.3722954 0.3343802
##
## Stationary point in original units:
##      Time      Temp
##  86.86148 176.67190
##
## Eigenanalysis:
## eigen() decomposition
## $values
## [1] -0.9233027 -1.3186949
##
## $vectors
##          [,1]       [,2]
## x1 -0.1601375 -0.9870947
## x2 -0.9870947  0.1601375
```

```
# The secondary point is a maximum (both eigenvalues are negative) and within the experimental d
esign range (no extrapolation)


# Also note that the block is significant, meaning that the processes shifted between the first
 set of data and the second.  This is not good.  The coefficient is -4.5, meaning the yield shif
ted down by 4.5% between the two blocks - a more significant effect than either temperatue or ti
me!  This is most easily seen by looking at the repeat center points.


# We can plot the fitted response as a contour plot.
contour(CR.rsm2, ~ x1 + x2, at = summary(CR.rsm2)$canonical$xs)
```




```
#OTHER example
library(rsm)
cube (2, n0 = 4)
```

```
##   run.order std.order x1.as.is x2.as.is
## 1         1         8        0        0
## 2         2         5        0        0
## 3         3         4        1        1
## 4         4         1       -1       -1
## 5         5         3       -1        1
## 6         6         6        0        0
## 7         7         7        0        0
## 8         8         2        1       -1
##
## Data are stored in coded form using these coding formulas ...
## x1 ~ x1.as.is
## x2 ~ x2.as.is
```

```
ccd.pick(k=2)
```

```
##    n.c n0.c blks.c n.s n0.s bbr.c wbr.s bbr.s  N alpha.rot alpha.orth
## 1    4    1      1   4    1     1     1     1 10  1.414214   1.414214
## 2    4    2      1   4    2     1     1     1 12  1.414214   1.414214
## 3    4    3      1   4    3     1     1     1 14  1.414214   1.414214
## 4    4    4      1   4    4     1     1     1 16  1.414214   1.414214
## 5    4    5      1   4    5     1     1     1 18  1.414214   1.414214
## 6    4    6      1   4    6     1     1     1 20  1.414214   1.414214
## 7    4    7      1   4    7     1     1     1 22  1.414214   1.414214
## 8    4    8      1   4    8     1     1     1 24  1.414214   1.414214
## 9    4    9      1   4    9     1     1     1 26  1.414214   1.414214
## 10   4   10      1   4   10     1     1     1 28  1.414214   1.414214
```

#Copy this matrix for example x1 x2 Time Temp y -1 -1 80 179 76.5 -1 1 80 180 77 1 0.1 90 179 78 1 1 90 180 79.5 0 0 85 175 79.9 0 0 85 175 80.3 0 0 85 175 80 0 0 85 175 79.7

```
#ccd2 <- read.table(file = "clipboard",  sep = "\t", header=TRUE)
#write.csv(ccd2, file = "ccd2.csv", row.names = FALSE)
ccd2<- read.csv("ccd2.csv", header= TRUE, sep=",")


ccd1<- as.coded.data(ccd2, x1 ~ (Time-85)/5, x2 ~ (Temp-175)/5)


Model_Y1<- rsm(y ~ SO(x1, x2), data = ccd1)
```

```
## Warning in rsm(y ~ SO(x1, x2), data = ccd1): Some coefficients are aliased - cannot use 'rsm'
methods.
##   Returning an 'lm' object.
```

```
Model_Y3<- rsm(y ~ FO(x1, x2) + PQ(x1, x2), data = ccd1)

par(mfrow = c(1, 2))
contour(Model_Y3, ~ x1+x2, image = TRUE, yagp=c(168,182, 2), xlabs=c("Time", "Temp"))
```

```
## Warning in plot.window(...): "yagp" is not a graphical parameter
```

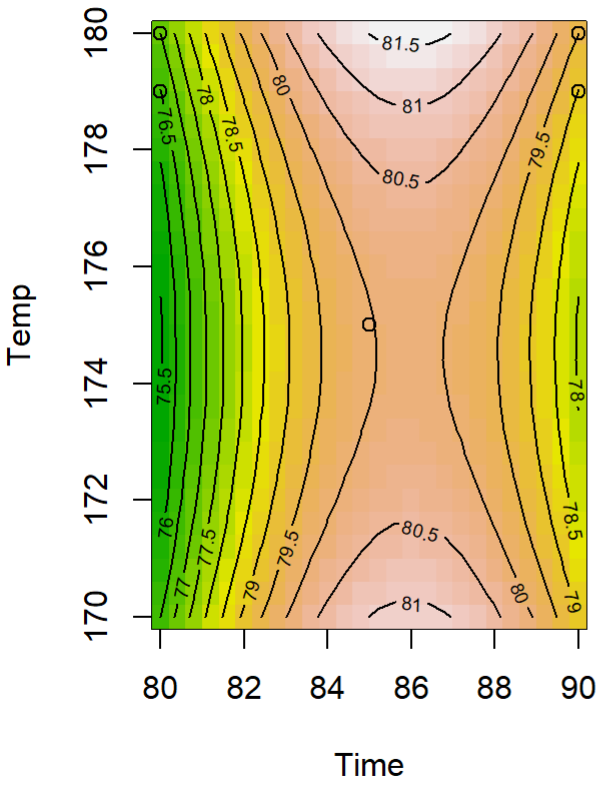
```
## Warning in plot.xy(xy, type, ...): "yagp" is not a graphical parameter
```

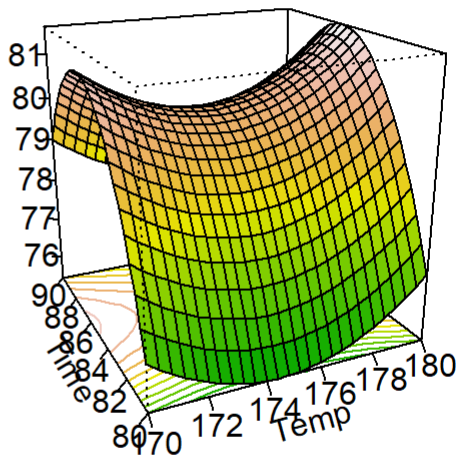
```
## Warning in axis(side = side, at = at, labels = labels, ...): "yagp" is not a
## graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "yagp" is not a
## graphical parameter
```

```
## Warning in box(...): "yagp" is not a graphical parameter
```

```
## Warning in title(...): "yagp" is not a graphical parameter
```

```
points(ccd1$Time, ccd1$Temp)
 persp(Model_Y3,  x1~x2, col = terrain.colors(50), contours = "colors")
```





```
max <- data.frame(x1 = 0.361, x2 = 0.257)
```

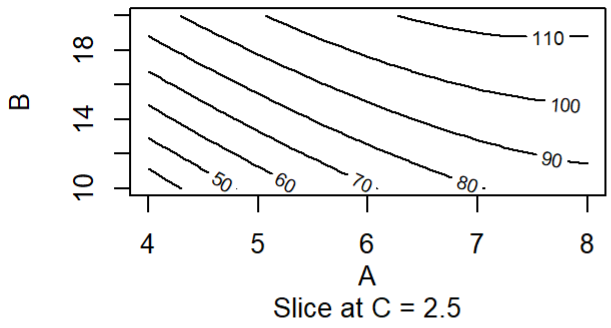
```
library(rsm)
ccd.pick(k=3)
```

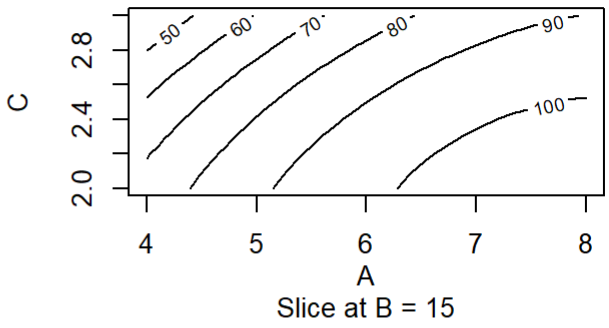
```
##    n.c n0.c blks.c n.s n0.s bbr.c wbr.s bbr.s  N alpha.rot alpha.orth
## 1    8    9      1   6    6     1     1     1 29  1.681793   1.680336
## 2    8    2      1   6    1     1     1     1 17  1.681793   1.673320
## 3    8    6      1   6    4     1     1     1 24  1.681793   1.690309
## 4    8    5      1   6    3     1     1     1 22  1.681793   1.664101
## 5    8   10      1   6    7     1     1     1 31  1.681793   1.699673
## 6    8    8      1   6    5     1     1     1 27  1.681793   1.658312
## 7    8    3      1   6    2     1     1     1 19  1.681793   1.705606
## 8    8    7      1   6    5     1     1     1 26  1.681793   1.712698
## 9    8    4      1   6    2     1     1     1 20  1.681793   1.632993
## 10   8    4      1   6    3     1     1     1 21  1.681793   1.732051
```

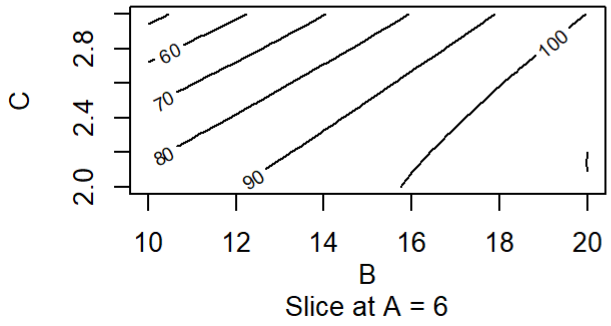
```
data(Treb)
treb.quad <- rsm(y ~ SO(x1, x2, x3), data = Treb)
summary(treb.quad)
```

```
##
## Call:
## rsm(formula = y ~ SO(x1, x2, x3), data = Treb)
##
##              Estimate Std. Error  t value  Pr(>|t|)
## (Intercept)  90.00000    1.16905  76.9859 7.006e-09 ***
## x1           19.75000    0.71589  27.5880 1.171e-06 ***
## x2           19.75000    0.71589  27.5880 1.171e-06 ***
## x3          -11.50000    0.71589 -16.0639 1.703e-05 ***
## x1:x2        -6.25000    1.01242  -6.1733 0.0016247 **
## x1:x3         4.75000    1.01242   4.6917 0.0053768 **
## x2:x3         6.75000    1.01242   6.6672 0.0011461 **
## x1^2         -9.37500    1.05376  -8.8967 0.0002986 ***
## x2^2         -1.37500    1.05376  -1.3048 0.2487686
## x3^2         -3.37500    1.05376  -3.2028 0.0239200 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.9975, Adjusted R-squared:  0.9929
## F-statistic: 218.9 on 9 and 5 DF,  p-value: 5.964e-06
##
## Analysis of Variance Table
##
## Response: y
##                 Df Sum Sq Mean Sq  F value    Pr(>F)
## FO(x1, x2, x3)   3 7299.0 2433.00 593.4146 8.448e-07
## TWI(x1, x2, x3)  3  428.8  142.92  34.8577 0.0008912
## PQ(x1, x2, x3)   3  351.5  117.16  28.5759 0.0014236
## Residuals        5   20.5    4.10
## Lack of fit      3   14.5    4.83   1.6111 0.4051312
## Pure error       2    6.0    3.00
##
## Stationary point of response surface:
##         x1         x2         x3
##  0.9236846 -1.7161183 -2.7698217
##
## Stationary point in original units:
##        A        B        C
## 7.847369 6.419409 1.115089
##
## Eigenanalysis:
## eigen() decomposition
## $values
## [1]   1.280298  -3.551452 -11.853845
##
## $vectors
##          [,1]       [,2]       [,3]
## x1 -0.1236692  0.5238084  0.8428112
## x2  0.8323200 -0.4077092  0.3755217
## x3  0.5403233  0.7479291 -0.3855551
```

```
par (mfrow=c(2,2))
contour(treb.quad, ~ x1+x2+x3 )
```

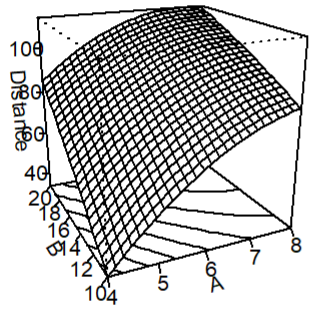




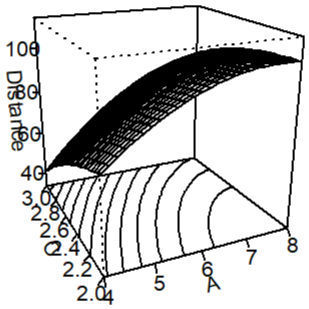


#3D response surface experiment

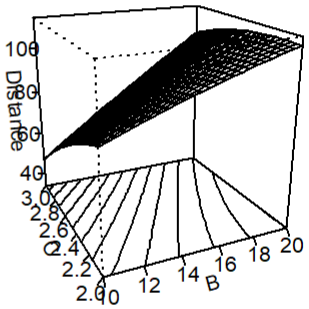
```
par (mfrow=c(1,3))
persp(treb.quad, ~ x1+x2+x3, zlab="Distance", contours=list(z="bottom"))
```



Slice at C = 2.5



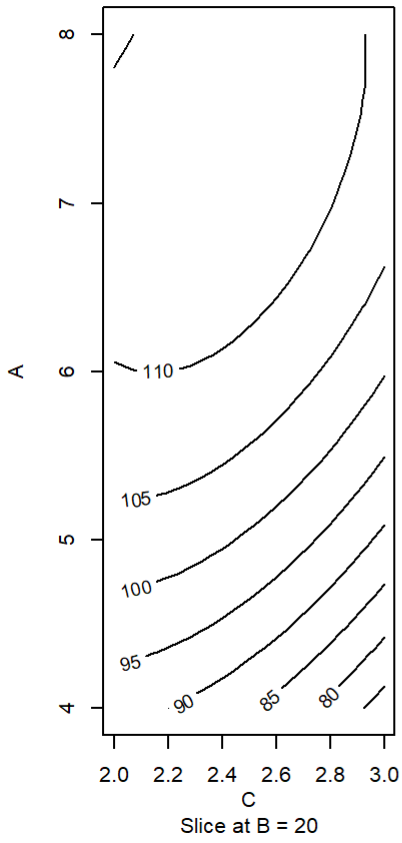
Slice at B = 15

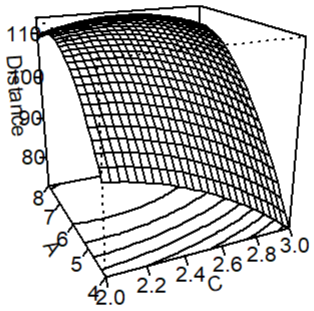


Slice at A = 6

#3D response surface experiment one factor

```
par (mfrow=c(1,3))
contour(treb.quad, x1~x3, at=list(x2=1))
persp(treb.quad, x1~x3, at=list(x2=1),zlab="Distance", contours=list(z="bottom"))
```





#DETERMINING OPTIMUM OPERATING CONDITIONS

```
ridge<-steepest(treb.quad, dist=seq(0, 1.412, by=.1), descent=FALSE)
```

```
## Path of steepest ascent from ridge analysis:
```

```
ridge
```

```
##    dist    x1    x2     x3 |     A      B      C |    yhat
## 1   0.0 0.000 0.000  0.000 | 6.000 15.000 2.5000 |  90.000
## 2   0.1 0.064 0.067 -0.037 | 6.128 15.335 2.4815 |  92.909
## 3   0.2 0.124 0.139 -0.073 | 6.248 15.695 2.4635 |  95.626
## 4   0.3 0.180 0.215 -0.105 | 6.360 16.075 2.4475 |  98.120
## 5   0.4 0.232 0.297 -0.134 | 6.464 16.485 2.4330 | 100.455
## 6   0.5 0.277 0.385 -0.158 | 6.554 16.925 2.4210 | 102.599
## 7   0.6 0.315 0.480 -0.175 | 6.630 17.400 2.4125 | 104.590
## 8   0.7 0.345 0.580 -0.185 | 6.690 17.900 2.4075 | 106.424
## 9   0.8 0.368 0.686 -0.185 | 6.736 18.430 2.4075 | 108.154
## 10  0.9 0.384 0.795 -0.177 | 6.768 18.975 2.4115 | 109.783
## 11  1.0 0.393 0.905 -0.161 | 6.786 19.525 2.4195 | 111.318
## 12  1.1 0.397 1.017 -0.137 | 6.794 20.085 2.4315 | 112.817
## 13  1.2 0.398 1.127 -0.107 | 6.796 20.635 2.4465 | 114.259
## 14  1.3 0.395 1.236 -0.073 | 6.790 21.180 2.4635 | 115.673
## 15  1.4 0.390 1.344 -0.034 | 6.780 21.720 2.4830 | 117.077
```

#MIXTURE EXPERIMENTS

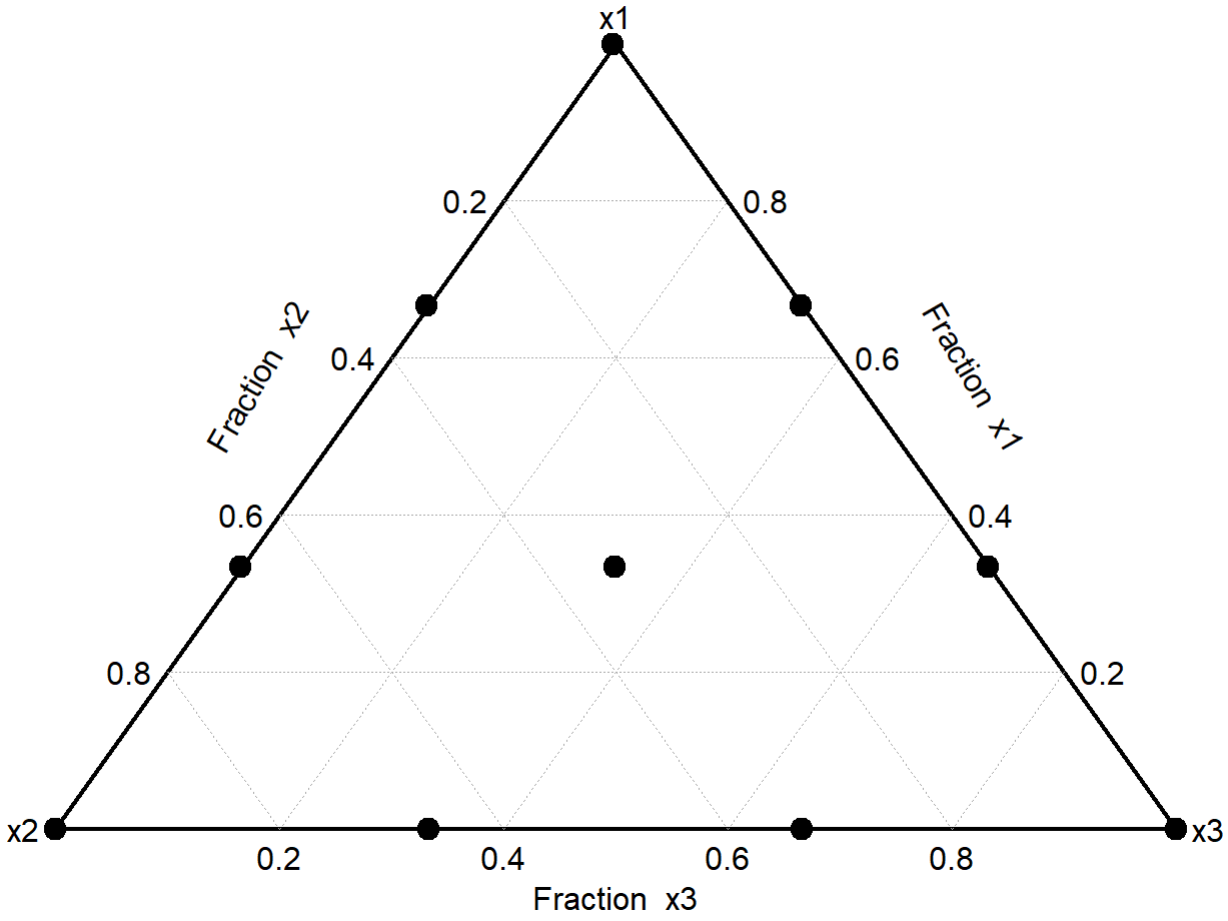
```
library(mixexp)
```

```
## Loading required package: lattice
```

```
SLD(3,2)
```

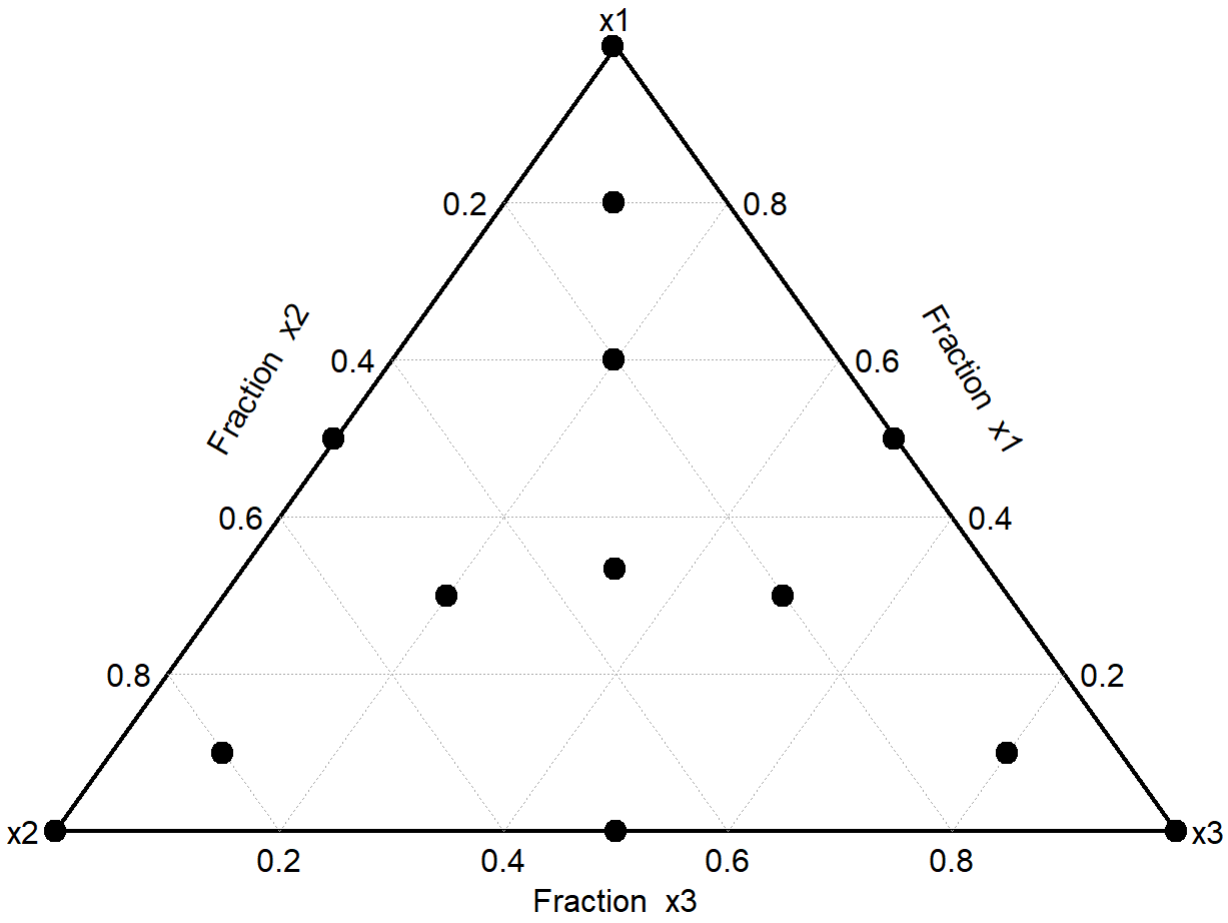
```
##    x1  x2  x3
## 1 1.0 0.0 0.0
## 2 0.5 0.5 0.0
## 3 0.0 1.0 0.0
## 4 0.5 0.0 0.5
## 5 0.0 0.5 0.5
## 6 0.0 0.0 1.0
```

```
des<-SLD(3,3)
DesignPoints(des)
```




```
library(daewr)
data(pest)
DesignPoints(pest)
```

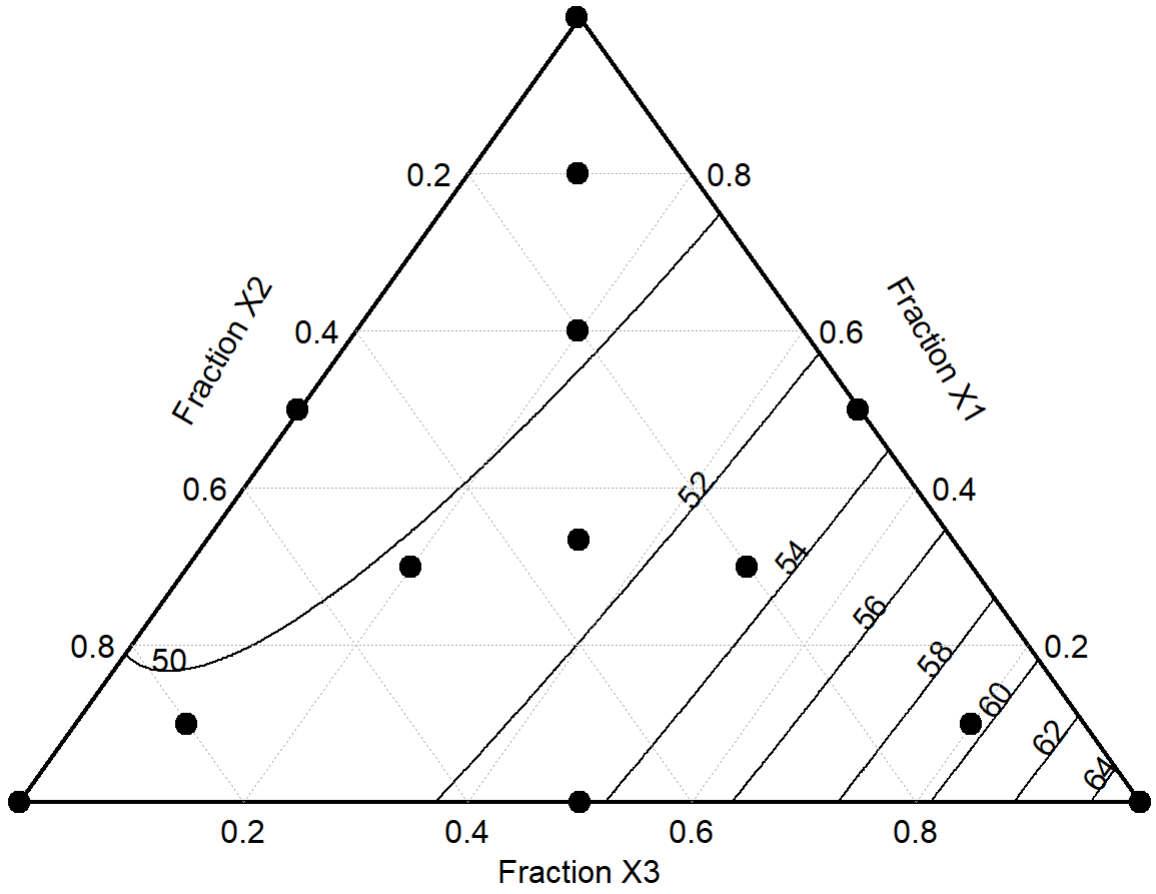
```
## Warning: the design matrix has more than three columns; the DesignPoints function
## only plots design points for designs with three mixture components. Component x1 is
## assumed to to be the first column of the design, x2 the second and x3 the third. Other
## columns are ignored. Use cornerlabs and axislabs to change variable names in the plot.
```



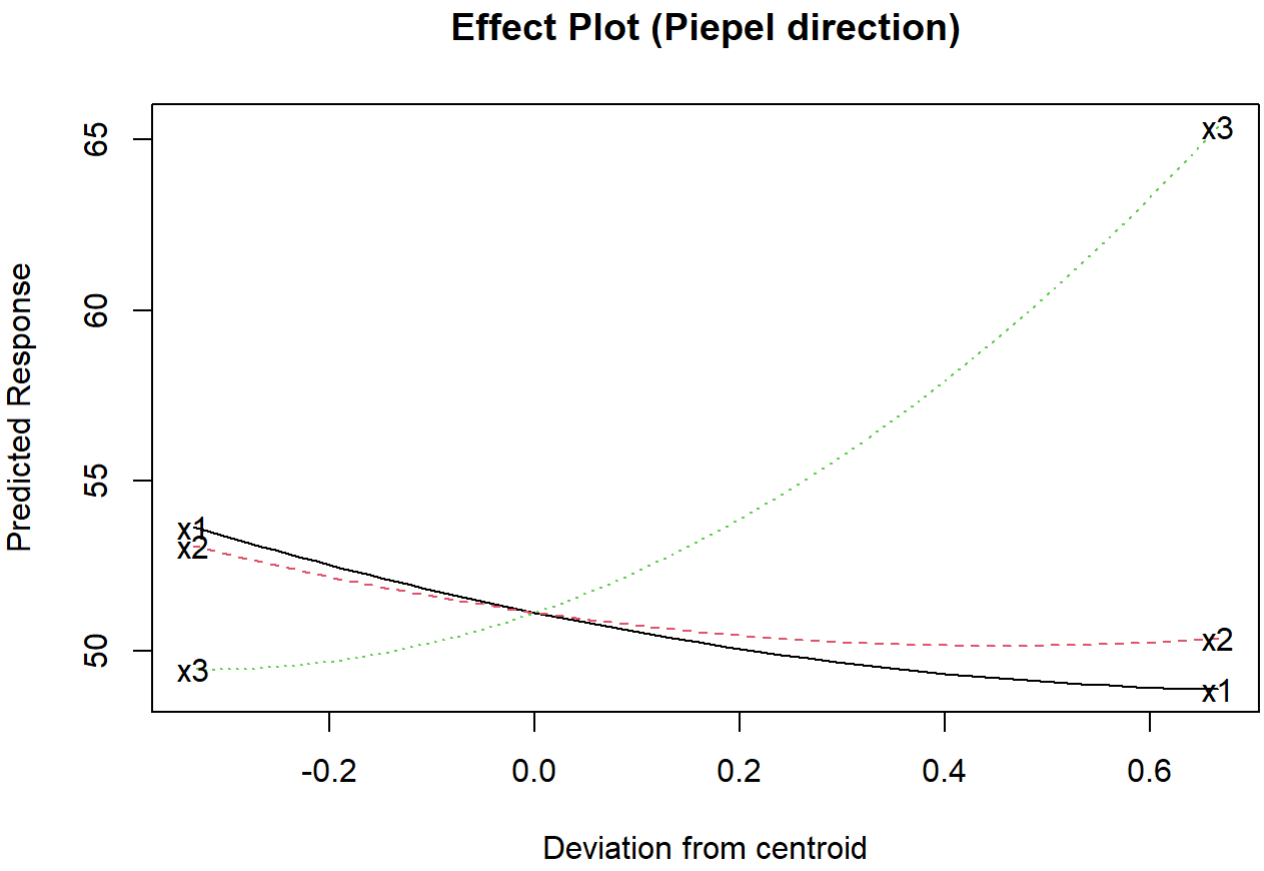

```
 spc <- lm(y ~ x1 + x2 + x3 + x1:x2 + x1:x3 + x2:x3 + x1:x2:x3 -1, data = pest)
summary(spc)
```

```
##
## Call:
## lm.default(formula = y ~ x1 + x2 + x3 + x1:x2 + x1:x3 + x2:x3 +
##     x1:x2:x3 - 1, data = pest)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.6134 -0.2553 -0.1051  0.2049  1.1253
##
## Coefficients:
##          Estimate Std. Error t value Pr(>|t|)
## x1        48.9056     0.5645  86.628 1.59e-10 ***
## x2        50.3951     0.5685  88.644 1.39e-10 ***
## x3        65.3870     0.5685 115.014 2.91e-11 ***
## x1:x2     -0.9156     3.0422  -0.301  0.77360
## x1:x3    -16.3642     3.0422  -5.379  0.00170 **
## x2:x3    -17.1440     3.0888  -5.550  0.00145 **
## x1:x2:x3   3.1052    18.0351   0.172  0.86896
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6317 on 6 degrees of freedom
## Multiple R-squared:  0.9999, Adjusted R-squared:  0.9999
## F-statistic: 1.299e+04 on 7 and 6 DF,  p-value: 4.146e-12
```

```
MixturePlot(des = pest,mod = 2)
```




```
EffPlot(des=pest,mod=2,dir=1)
```


Effect Plot (Piepel direction)
Predicted Response
65
60
55
50
x3
x1
x2
x2
x3
x1
-0.2
0.0
0.2
0.4
0.6
Deviation from centroid

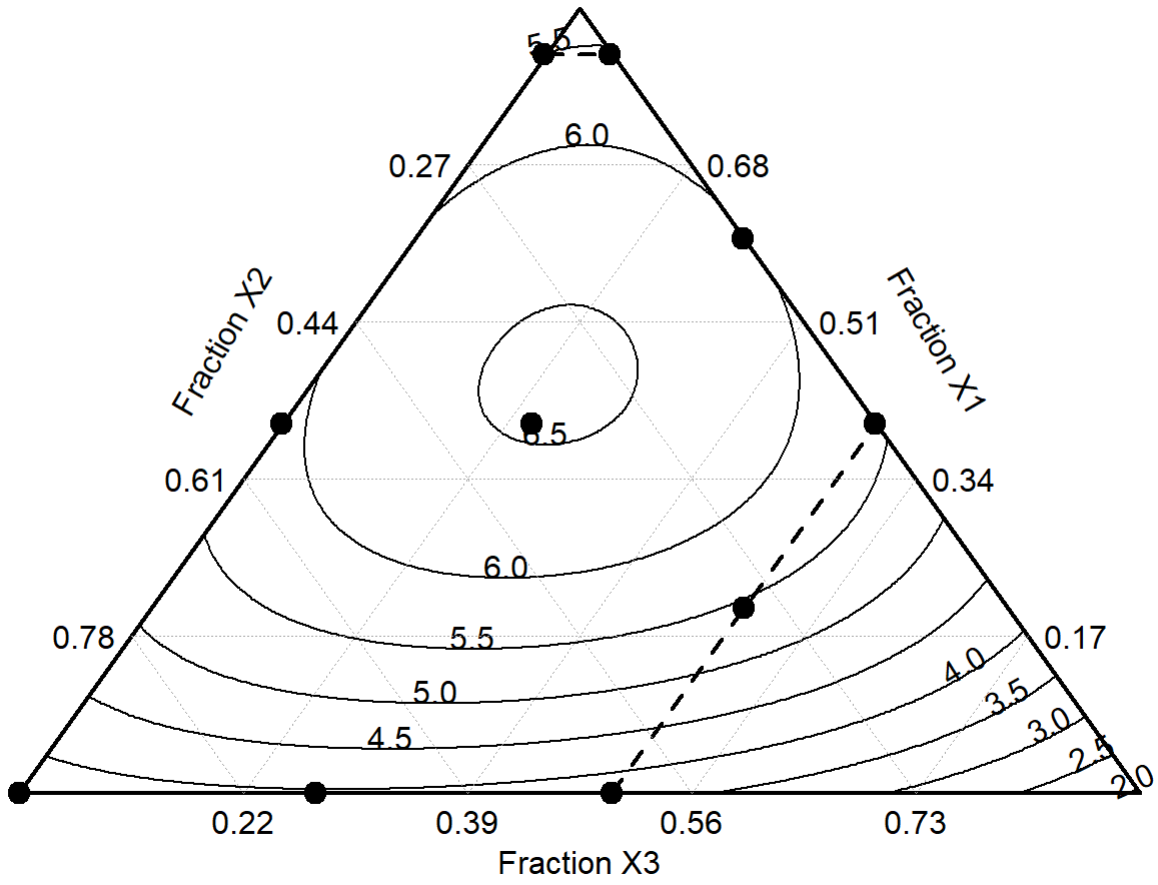
```
##             [,1]     [,2]       [,3]     [,4]       [,5]     [,6]
##  [1,]  0.66666666 48.89339  0.66666666 50.38323  0.66666666 65.37513
##  [2,]  0.63999999 48.91227  0.63999999 50.33221  0.63999999 64.52669
##  [3,]  0.61333332 48.93707  0.61333332 50.28796  0.61333332 63.70150
##  [4,]  0.58666666 48.96778  0.58666666 50.25048  0.58666666 62.89956
##  [5,]  0.55999999 49.00440  0.55999999 50.21978  0.55999999 62.12087
##  [6,]  0.53333333 49.04694  0.53333333 50.19585  0.53333333 61.36544
##  [7,]  0.50666666 49.09540  0.50666666 50.17870  0.50666666 60.63326
##  [8,]  0.47999999 49.14977  0.47999999 50.16832  0.47999999 59.92434
##  [9,]  0.45333333 49.21005  0.45333333 50.16472  0.45333333 59.23867
## [10,]  0.42666666 49.27626  0.42666666 50.16789  0.42666666 58.57625
## [11,]  0.39999999 49.34837  0.39999999 50.17783  0.39999999 57.93708
## [12,]  0.37333333 49.42640  0.37333333 50.19455  0.37333333 57.32117
## [13,]  0.34666666 49.51035  0.34666666 50.21804  0.34666666 56.72850
## [14,]  0.32000000 49.60021  0.32000000 50.24831  0.32000000 56.15910
## [15,]  0.29333333 49.69599  0.29333333 50.28535  0.29333333 55.61294
## [16,]  0.26666666 49.79768  0.26666666 50.32917  0.26666666 55.09004
## [17,]  0.24000000 49.90528  0.24000000 50.37976  0.24000000 54.59039
## [18,]  0.21333333 50.01881  0.21333333 50.43713  0.21333333 54.11399
## [19,]  0.18666666 50.13824  0.18666666 50.50127  0.18666666 53.66085
## [20,]  0.16000000 50.26360  0.16000000 50.57218  0.16000000 53.23096
## [21,]  0.13333333 50.39486  0.13333333 50.64987  0.13333333 52.82432
## [22,]  0.10666667 50.53204  0.10666667 50.73433  0.10666667 52.44093
## [23,]  0.08000000 50.67514  0.08000000 50.82557  0.08000000 52.08080
## [24,]  0.05333333 50.82415  0.05333333 50.92358  0.05333333 51.74392
## [25,]  0.02666667 50.97908  0.02666667 51.02836  0.02666667 51.43030
## [26,]  0.00000000 51.13992  0.00000000 51.13992  0.00000000 51.13992
## [27,] -0.01333333 51.22256 -0.01333333 51.19824 -0.01333333 51.00346
## [28,] -0.02666667 51.30668 -0.02666667 51.25826 -0.02666667 50.87280
## [29,] -0.04000000 51.39228 -0.04000000 51.31997 -0.04000000 50.74796
## [30,] -0.05333333 51.47935 -0.05333333 51.38337 -0.05333333 50.62894
## [31,] -0.06666667 51.56791 -0.06666667 51.44846 -0.06666667 50.51572
## [32,] -0.08000000 51.65794 -0.08000000 51.51525 -0.08000000 50.40832
## [33,] -0.09333334 51.74945 -0.09333334 51.58373 -0.09333334 50.30673
## [34,] -0.10666667 51.84244 -0.10666667 51.65391 -0.10666667 50.21096
## [35,] -0.12000000 51.93691 -0.12000000 51.72578 -0.12000000 50.12100
## [36,] -0.13333334 52.03286 -0.13333334 51.79934 -0.13333334 50.03685
## [37,] -0.14666667 52.13029 -0.14666667 51.87460 -0.14666667 49.95852
## [38,] -0.16000000 52.22920 -0.16000000 51.95155 -0.16000000 49.88599
## [39,] -0.17333334 52.32958 -0.17333334 52.03019 -0.17333334 49.81929
## [40,] -0.18666667 52.43144 -0.18666667 52.11053 -0.18666667 49.75839
## [41,] -0.20000001 52.53479 -0.20000001 52.19256 -0.20000001 49.70331
## [42,] -0.21333334 52.63961 -0.21333334 52.27628 -0.21333334 49.65404
## [43,] -0.22666667 52.74591 -0.22666667 52.36170 -0.22666667 49.61058
## [44,] -0.24000001 52.85369 -0.24000001 52.44881 -0.24000001 49.57294
## [45,] -0.25333334 52.96294 -0.25333334 52.53761 -0.25333334 49.54111
## [46,] -0.26666667 53.07368 -0.26666667 52.62811 -0.26666667 49.51510
## [47,] -0.28000001 53.18589 -0.28000001 52.72031 -0.28000001 49.49490
## [48,] -0.29333334 53.29959 -0.29333334 52.81419 -0.29333334 49.48051
## [49,] -0.30666668 53.41476 -0.30666668 52.90977 -0.30666668 49.47193
## [50,] -0.32000001 53.53141 -0.32000001 53.00704 -0.32000001 49.46917
## [51,] -0.33333334 53.64954 -0.33333334 53.10601 -0.33333334 49.47222
```

#Other example of ANALYSIS OF MIXTURE EXPERIMENTS

```
data(polvdat)
sqm <- lm(y ~ x1 + x2 + x3 + x1:x2 + x1:x3 + x2:x3 + x1:x2:x3 -1, data = polvdat)
summary(sqm)
```

```
##
## Call:
## lm.default(formula = y ~ x1 + x2 + x3 + x1:x2 + x1:x3 + x2:x3 +
##     x1:x2:x3 - 1, data = polvdat)
##
## Residuals:
##        1        2        3        4        5        6        7        8
## -0.17957 -0.02142  0.03359 -0.12009  0.14423 -0.20166  0.09631 -0.14312
##        9       10       11       12
##  0.20803  0.01123  0.29123 -0.11877
##
## Coefficients:
##          Estimate Std. Error t value Pr(>|t|)
## x1         4.4259     0.4483   9.873 0.000182 ***
## x2         3.5181     0.3079  11.427 8.99e-05 ***
## x3         1.2367     1.6150   0.766 0.478400
## x1:x2      6.9004     2.0179   3.420 0.018846 *
## x1:x3      8.9528     4.1427   2.161 0.083071 .
## x2:x3      5.3135     3.4988   1.519 0.189310
## x1:x2:x3  25.5460    11.2023   2.280 0.071499 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2374 on 5 degrees of freedom
## Multiple R-squared:  0.9992, Adjusted R-squared:  0.9981
## F-statistic: 920.9 on 7 and 5 DF,  p-value: 1.815e-07
```

```
MixturePlot(des = polvdat, mod = 4, lims=c(0,.8,.1,.95, .05, .50), constrts=TRUE, pseudo=TRUE)
```


5.5
6.0
0.27
0.68
Fraction X2
0.44
0.51
Fraction X1
6.5
0.61
0.34
6.0
0.78
5.5
0.17
5.0
4.0
3.5
4.5
3.0
2.5
2.0
0.22
0.39
0.56
0.73
Fraction X3