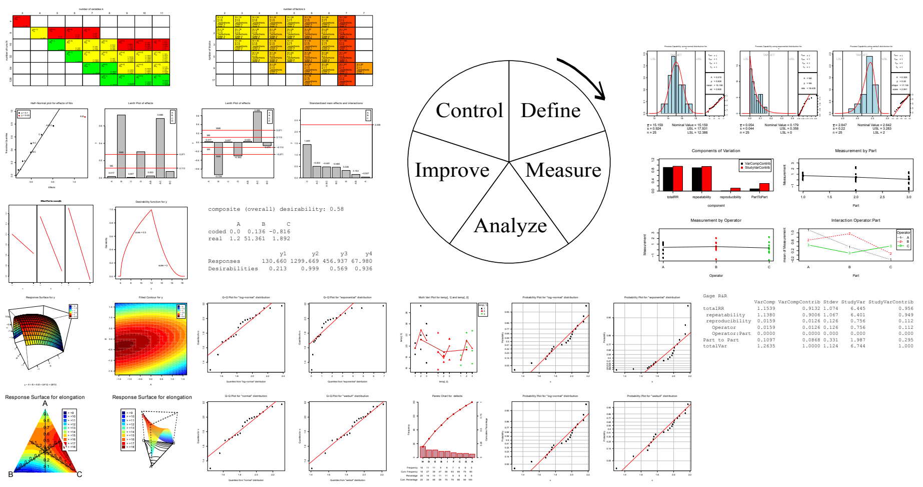


```
composite (overall) desirability: 0.58

         A      B      C
coded 0.0  0.136 -0.816
real  1.2 51.361  1.892

                    y1       y2      y3     y4
Responses      130.660 1299.669 456.937 67.980
Desirabilities   0.213    0.999   0.569  0.936
```

```
Gage R&R
                   VarComp VarCompContrib Stdev StudyVar StudyVarContrib
totalRR             1.1539         0.9132 1.074    6.445           0.956
 repeatability      1.1380         0.9006 1.067    6.401           0.949
 reproducibility    0.0159         0.0126 0.126    0.756           0.112
   Operator         0.0159         0.0126 0.126    0.756           0.112
   Operator:Part    0.0000         0.0000 0.000    0.000           0.000
Part to Part        0.1097         0.0868 0.331    1.987           0.295
totalVar            1.2635         1.0000 1.124    6.744           1.000
```

Working with the qualityTools package

A short introduction

Thomas Roth

November 5, 2010

This vignette is intended to give a short introduction into the methods of the qualityTools package. The qualityTools package contains methods associated with the **D**efine **M**easure **A**nalyze **I**mprove and **C**ontrol (i.e. **DMAIC**) problem solving cycle of the Six Sigma Quality Management methodology. Usage of these methods is illustrated with the help of artificially created datasets.

- **Define:** Pareto Chart
- **Measure:** Probability and Quantile-Quantile Plots, Process Capability Ratios for various distributions and Gage R&R
- **Analyze:** Pareto Chart, Multi-Vari Chart, Dot Plot
- **Improve:** Full and fractional factorial, response surface, mixture and taguchi designs as well as the desirability approach for simultaneous optimization of more than one response variable. Normal, Pareto and Lenth Plot of effects as well as Interaction Plots

Contents

1. Working with the qualityTools package

Working with the qualityTools package is straightforward as you will see in the next few pages. The qualityTools package was implemented for teaching purposes in order to serve as a (Six-Sigma)-Toolbox and contains methods that are associated to a problem solving cycle. There are many problem solving cycles around with new ones emerging although most of these new ones take on special aspects. A very popular problem solving cycle is the **PDCA** cycle (i.e. plan ↷ do↷ check↷ act↻) which was made popular by Deming[1] but goes back to Shewart[2]. As part of the widely known and accepted Six-Sigma-Methodology some enhancements to this problem solving cycle were made and a problem solving cycle consisting of the five phases Define, Measure, Analyze, Improve and Control emerged.

Define Describe the problem and its (financial) consequences. Interdisciplinary work-groups contribute to the problem and its consequences which is the pivotal stage in narrowing down the problem. Process flow diagrams identify crucial process elements (i.e. activities), creativity techniques such as Brainwriting and Brainstorming as well as the SIPOC[3] technique should lead, depending on the future size of the project, to possibly a project charter.

Measure Come up with a reasonable plan for collecting the required data and make sure that the measurement systems are capable (i.e. no or known bias and as little system immanent variation contributing to the measurements as possible). **Variation** and **bias** are the enemy to finding effects. The bigger the background noise the less probable are the chances of success using limited resources for all kinds of experiments.

Analyze Try to find the root causes of the problem using various statistical methods such as histograms, regression, correlation, distribution identification, analysis of variance, multi-vari-charts.

Improve Use designed experiments i.e. full and fractional factorials, response surface designs, mixture designs, taguchi designs and the desirability concept to find optimal settings or solutions for a problem.

Control Once an improvement was achieved it needs to be secured, meaning arrangements need to be implemented in order to secure the level of improvement. Besides proper documentation, the use of statistical process control (i.e. quality-control-charts) can be used to monitor the behavior of a process. Although quite often referred to as **Show Programm for Customers**, SPC is able to help to distinguish between **common causes** and **special causes** in the process behavior.

[1]William E. Deming
[2]Walter A. Shewhart
[3]Suppliers, inputs, process, outputs, customers

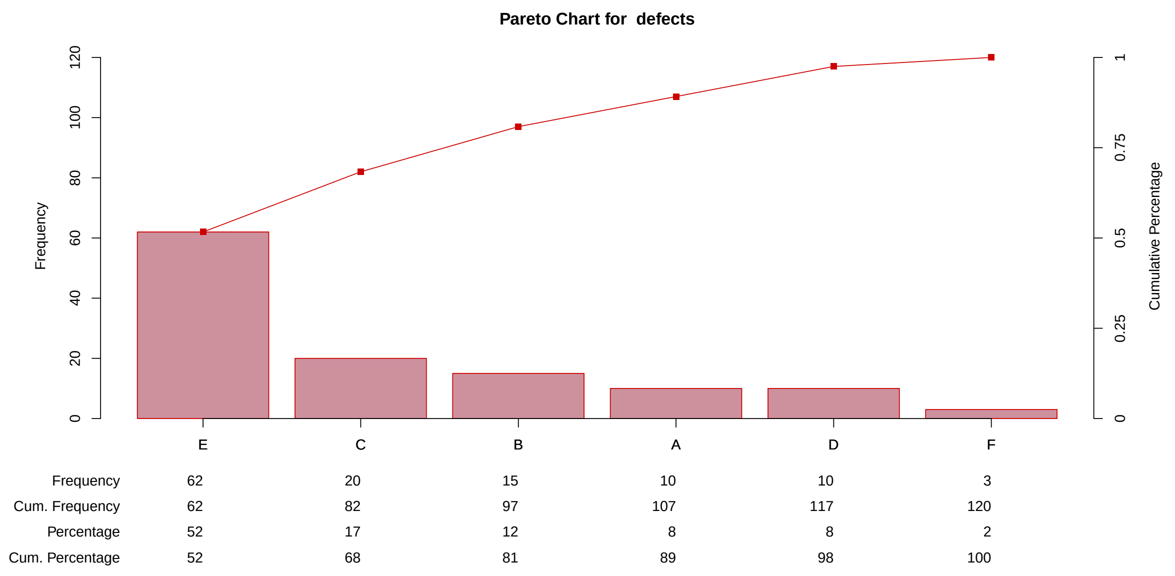
2. qualityTools in DEFINE

Most techniques in the Define phase are not related to substantial use of statistical methods. The objective of the DEFINE phase is to bring together all parties concerned, grasp their knowledge and insights to the process involved, set a common objective and DEFINE how each party contributes(or the role each party takes) to the solving of the problem. In order not to get lost in subsequent meetings and ongoing discussion, this common objective, the contribution of each party, milestones and responsibilities need to be written down in what is known to be a Project Charter. Of course, problems with easy-to-identify causes are not subject of these kind of projects.

However, a classical visualization technique that is used in this phase and available in the qualityTools package is the pareto chart. Pareto charts are special forms of bar charts that help to separate the vital few from the trivial many causes for a given problem (e.g.the most frequent cause for a defective product). This way pareto charts visualize how much a cause contributes to a specific issue.

Suppose a company is investigating non compliant units (products). 120 units were investigated and 6 different types of defects (qualitative data) were found. The defects are named A to F. The defects data can be found in defects.

```
#create artificial defect data set
> defects = c(rep("E", 62), rep("B", 15), rep("F", 3), rep("A", 10), rep
   ("C",20), rep("D", 10))
> paretoChart(defects)
```



Frequency	62	20	15	10	10	3
Cum. Frequency	62	82	97	107	117	120
Percentage	52	17	12	8	8	2
Cum. Percentage	52	68	81	89	98	100

Figure 1: Pareto Chart

This pareto chart might convey the message that in order to solve 68 percent of the problem 33 percent of the causes (**vital few**[4]) need to be subject of an investigation.

Besides this use case, pareto charts are also used for visualizing the effect sizes of different factors for designed experiments (see `paretoPlot`).

[4]the vital few and the trivial many - 20 percent of the defects cause 80 percent of the problems

3. qualityTools in MEASURE

Collecting data involves the use of measurement systems often referred to as gages. In order to make a statement regarding the quality, i.e. the degree in which a set of inherent characteristics meets requirements, of a product[1], the capability of the measurement system used needs to be validated.

Gages can have two types of impairments:

- a bias (an assumed constant shift of values for measurements of equal magnitude)
- variation
  - introduced by other factors e.g. operators using these gages
  - system immanent variation of the measurement system itself

These impairments lead to varying measurements for repeated measurements of the same unit (e.g. a product). The amount of tolerable variation of course depends on the number of distinctive categories you need to be able to identify in order to characterize the product. This tolerable amount of variation for a measurement system relates directly to the tolerance range of the characteristics of a product.

3.1. Gage Repeatability&Reproducibility

A common procedure applied in industry is to perform a Gage R&R analysis to assess the repeatability and the reproducibility of a measurement system. R&R stands for repeatability and reproducibility. Repeatability hereby refers to the precision of a measurement system (i.e. the standard deviation of subsequent measurements of the same unit). Reproducibility is the part of the overall variance that models the effect of different e.g. operators performing measurements on the same unit and a possible interaction between different operators and parts measured within this Gage R&R. The overall model is given by

$$\sigma^2_{total} = \sigma^2_{Parts} + \sigma^2_{Operator} + \sigma^2_{Parts \times Operator} + \sigma^2_{Error} \quad (1)$$

where σ^2_{Parts} models the variation between different units of the same process. σ^2_{Parts} is thus an estimate of the inherent process variability. Repeatability is modeled by σ^2_{Error} and reproducibility by $\sigma^2_{Operator} + \sigma^2_{Parts \times Operator}$.

Suppose 9 randomly chosen units were measured by 3 randomly chosen operators. Each operator measured each unit three times in a randomly chosen order. The units were presented in a way they could not be distinguished by the operators.

The corresponding gage R&R design can be created using the `gageRRDesign` method of the qualityTools package. The measurements are assigned to this design using the response method. Methods for analyzing this design are given by `gageRR` and `plot`.

```
#set start for random number generation
> set.seed(1234)
#create a gage RnR design
> design  = gageRRDesign(Operators = 3, Parts = 9, Measurements = 3)
#set the response
> response(design) = rnorm(3*9*3, mean = 20)
#perform Gage RnR
> gdo = gageRR(design)
```

```
AnOVa Table -  crossed Design
               Df Sum Sq Mean Sq F value Pr(>F)
Operator        2  1.456 0.72779  0.6325 0.5351
Part            8 11.696 1.46196  1.2706 0.2782
Operator:Part  16 11.937 0.74607  0.6484 0.8293
Residuals      54 62.131 1.15058

----------
AnOVa Table Without Interaction -  crossed Design
             Df Sum Sq Mean Sq F value Pr(>F)
Operator      2  1.456 0.72779  0.6878 0.5060
Part          8 11.696 1.46196  1.3817 0.2198
Residuals    70 74.068 1.05812

----------

Gage R&R
                 VarComp VarCompContrib Stdev StudyVar StudyVarContrib
totalRR           1.0851         0.8698 1.042    6.250           0.933
 repeatability    1.0581         0.8482 1.029    6.172           0.921
 reproducibility  0.0270         0.0216 0.164    0.985           0.147
   Operator       0.0270         0.0216 0.164    0.985           0.147
   Operator:Part  0.0000         0.0000 0.000    0.000           0.000
Part to Part      0.1624         0.1302 0.403    2.418           0.361
totalVar          1.2475         1.0000 1.117    6.702           1.000

---
 * Contrib equals Contribution in %

#visualization of Gage RnR
>plot(gdo)
```

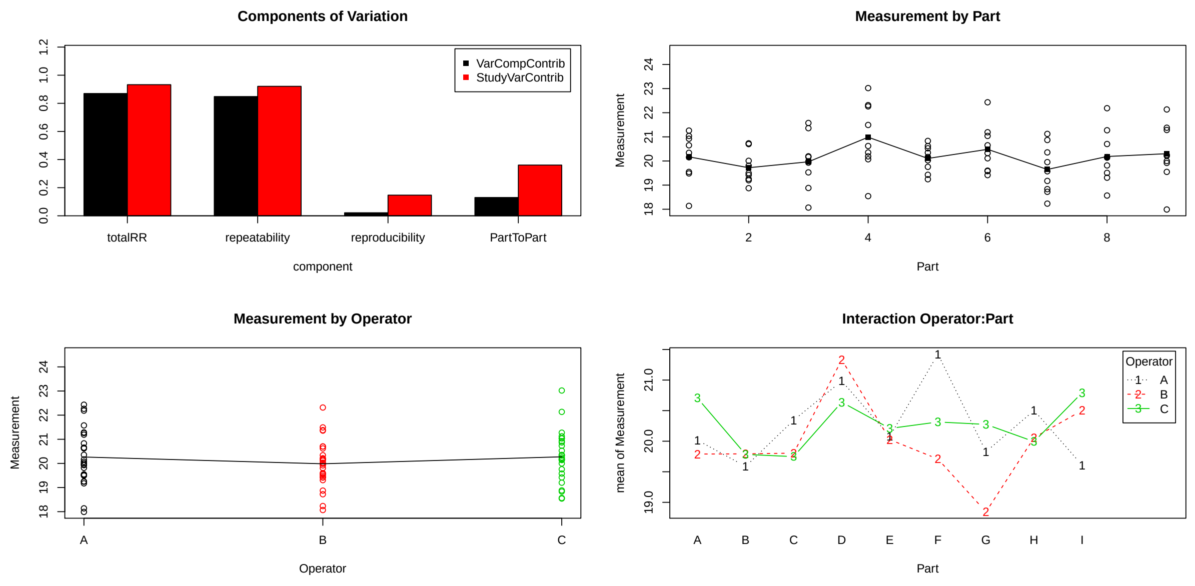


Figure 2: Visualization of the Gage R&R with random data

The standard graphical output of a Gage R&R is given in figure 2. A complete explanation of this output is beyond the scope of this vignette.

4. qualityTools in ANALYZE

4.1. Process Capability

Besides the capability of a measurement system, often the capability of a process is of interest or needs to be assessed e.g. as part of a supplier customer relationship in industry. Process Capability Indices basically tells one how much of the tolerance range is being used by common cause variation of the considered process. Using these techniques one can state how many units (e. g. products) are expected to fall outside the tolerance range (i.e. defective regarding the requirements determined before) if for instance production continues without intervention. It also gives insights into where to center the process if shifting is possible and meaningful in terms of costs. There are three indices which are also defined in the corresponding ISO 21747:2006 document[1] .

$$c_p = \frac{USL - LSL}{Q_{0.99865} - Q_{0.00135}} \tag{2}$$

$$c_{pkL} = \frac{Q_{0.5} - LSL}{Q_{0.5} - Q_{0.00135}} \tag{3}$$

$$c_{pkU} = \frac{USL - Q_{0.5}}{Q_{0.5} - Q_{0.00135}} \tag{4}$$

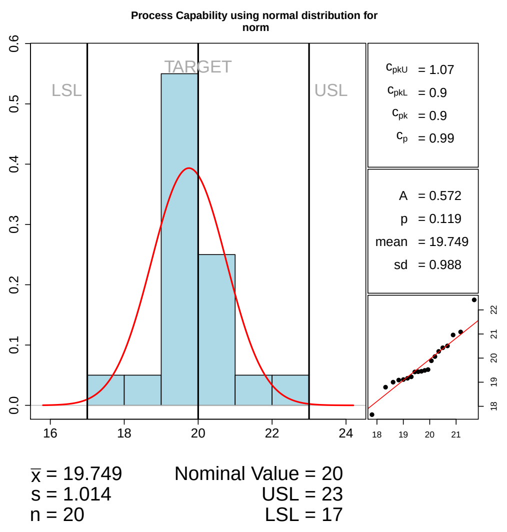
c_p is the potential process capability giving one the process capability that could be achieved if the process can be centered within specification limits[5] and c_{pk} is the actual process capability which incorporates the location of the distribution (i.e. the center) of the characteristic within the specification limits. For one sided specification limits c_{pkL} and c_{pkU} exist with c_{pk} being equal to the smallest capability index. As one can imagine in addition the location of the distribution of the characteristic the shape of the distribution is relevant too. Assessing the fit of a specific distribution for given data can be done via probability plots (`ppPlot`) and quantile-quantile plots (`qqPlot`), as well as formal test methods like the Anderson Darling Test.

Process capabilities can be calculated with the `pcr` method of the qualityTools package. The `pcr` method plots a histogram of the data, the fitted distribution and returns the capability indices along with the estimated[6] parameters of the distribution, an Anderson Darling Test for the specified distribution and the corresponding QQ-Plot.

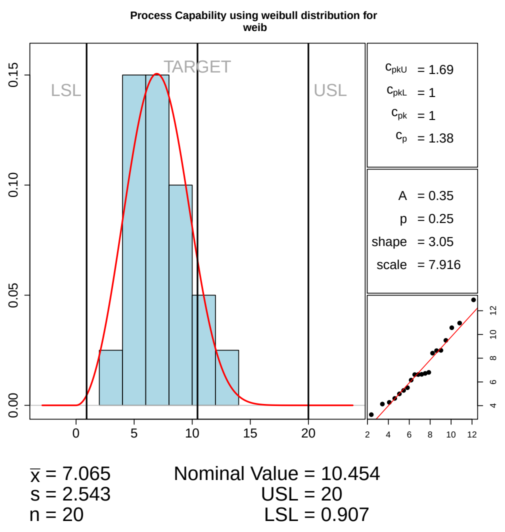
```
> set.seed(1234)
#generate some data
> norm = rnorm(20, mean = 20)
#generate some data
> weib = rweibull(20, shape = 2, scale = 8)
#process capability
> pcr(norm, "normal", lsl = 17, usl = 23)
#process cabapility
> pcr(weib, "weibull", usl = 20)
```

[5]USL - Upper Specification Limit
LSL - Lower Specification Limit

[6]Fitting the distribution itself is accomplished by the fitdistr method of the R-package MASS.



(a) normal distribution



(b) weibull distribution

Figure 3: Process Capability Ratios for weibull and normal distribution

Along with the graphical representation an Anderson Darling Test for the corresponding distribution is returned.

```
        Anderson Darling Test for weibull distribution

data:  x[, 1]
A = 0.3505, shape = 3.050, scale = 7.916, p-value > 0.25
alternative hypothesis: true distribution is not equal to weibull
```

Q-Q Plots can be calculated with the `qqPlot` function of the qualityTools package (figure 4).

```
> par(mfrow = c(1,2))
> qqPlot(weib, "weibull"); qqPlot(weib, "normal")
```

Probability Plots can be calculated with the `ppPlot` function of the qualityTools package (figure 5).

```
> ppPlot(norm, "weibull"); ppPlot(norm, "normal")
```

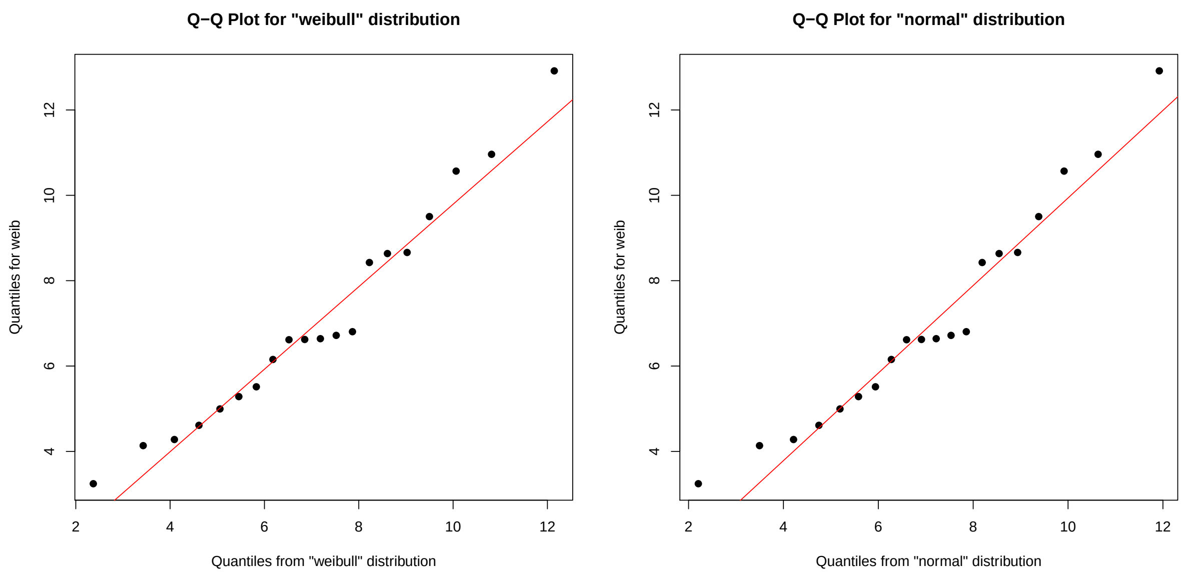


Figure 4: QQ-Plots for different distributions

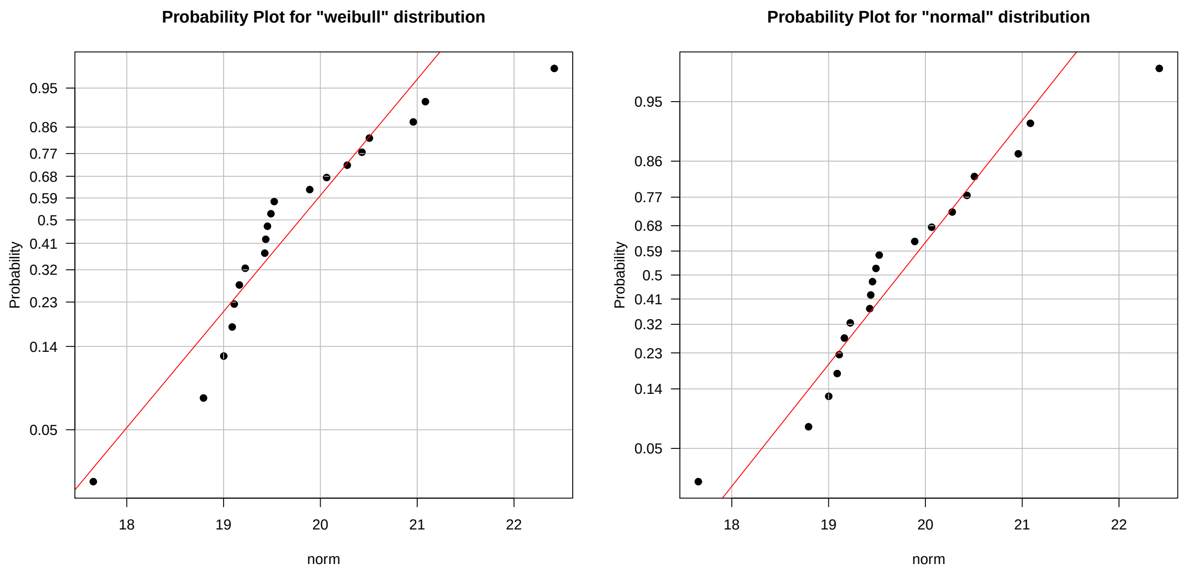


Figure 5: PP-Plots for different distributions

5. qualityTools in IMPROVE

Each process has a purpose. The effectiveness of a process can be expressed with the help of (quality) characteristics. Those characteristics can be denoted as the **responses** of a process. In order to attain the desired values for the responses certain settings need to be arranged for the process. Those settings refer to the input variables of the process. Working with designed experiments it is helpful to refer to the (black box) process model (figure 6).

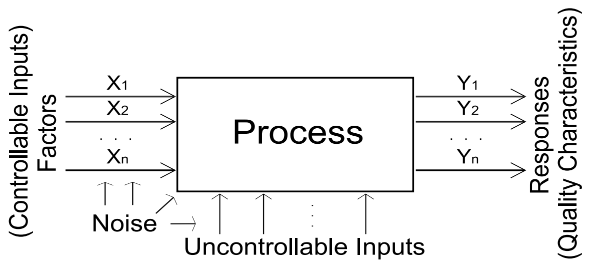


Figure 6: Black Box model of a process

In general input variables can be distinguished into controllable and disturbance variables. Input variables that can be controlled and have an assumed effect on the responses are denoted as **factors**. Input variables that are not factors are either hard to change (e.g. the hydraulic fluid in a machine) or varying them does not make good economic sense (e.g. the temperature or humidity in a factory building). These hard-to-change factors are also called uncontrollable input variables. It is attempted to held those variables constant. Disturbance variables affect the outcomes of a process by introducing noise such as small variations in the controllable and uncontrollable input variables which leads to variations in the response variables despite identical factor settings in an experiment.

5.1. 2^k Factorial Designs

In order to find more about this black box model one can come up with a 2^k factorial design by using the method `facDesign` of the qualityTools package. As used in textbooks k denotes the number of factors. A design with k factors and 2 combinations per factor gives you 2^k different factor combinations and thus what is called runs.

Suppose a process has 5 factors A, B, C, D and E. The yield (i.e. response) of the process is measured in percent. Three of the five factors are assumed by the engineers to be relevant to the yield of the process. These three factors are to be named Factor 1, Factor 2 and Factor 3 (A, B and C). The (unknown relations of the factors of the) process (are) is simulated by the method `simProc` of the qualityTools package. Factor 1 is to be varied from 80 to 120, factor B from 120 to 140 and factor C from 1 to 2 . Low factor settings are assigned a -1 and high values a +1.

```
> set.seed(1234)
#fdo - factorial design object
> fdo = facDesign(k = 3, centerCube = 4)
#optional
> names(fdo) = c("Factor␣1", "Factor␣2", "Factor␣3")
```

```
#optional
> lows(fdo) = c(80, 120, 1)
#optional
> highs(fdo) = c(120, 140, 2)
#information about the factorial design
> summary(fdo)

Information about the factors:

            A        B        C
low        80      120        1
high      120      140        2
name Factor 1 Factor 2 Factor 3
unit
type  numeric  numeric  numeric

-----------
   StandOrd RunOrder Block  A  B  C  y
1         1        1     1 -1 -1 -1 NA
6         6        2     1  1 -1  1 NA
8         8        3     1  1  1  1 NA
3         3        4     1 -1  1 -1 NA
2         2        5     1  1 -1 -1 NA
4         4        6     1  1  1 -1 NA
5         5        7     1 -1 -1  1 NA
7         7        8     1 -1  1  1 NA
9         9        9     1  0  0  0 NA
10       10       10     1  0  0  0 NA
11       11       11     1  0  0  0 NA
12       12       12     1  0  0  0 NA
```

The response of this fictional process is given by the simProc method of the qualityTools package. The yield for Factor 1, Factor 2 and Factor 3 taking values of 80, 120 and 1 can be calculated using

```
#set first value
> yield = simProc(x1 = 80, x2 = 120, x3 = 1)
```

Setting all the yield of this artificial black box process gives a very long line of R-Code.

```
> yield = c(simProc(80,120, 1),simProc(120,120, 2),simProc(120,140, 2),
   simProc(80,140, 1),simProc(120,120, 1),simProc(120,140, 1),simProc
   (80,120, 2),simProc(80,140, 2), simProc(90,130, 1.5), simProc(90,130,
    1.5), simProc(90,130, 1.5), simProc(90,130, 1.5))
```

Assigning the yield to the factorial design can be done using the response method.

```
>response(fdo) = yield  #assign yield to the factorial design object
```

Analyzing this design is quite easy using the methods effectPlot, interactionPlot, lm as well as wirePlot and contourPlot (figure 7)

```
> effectPlot(fdo)
> interactionPlot(fdo)
```

The factorial design in fdo can be handed without any further operations directly to the base lm method of R.

```
> lm.1 = lm(yield ~ A*B*C, data = fdo)
> summary(lm.1)
```

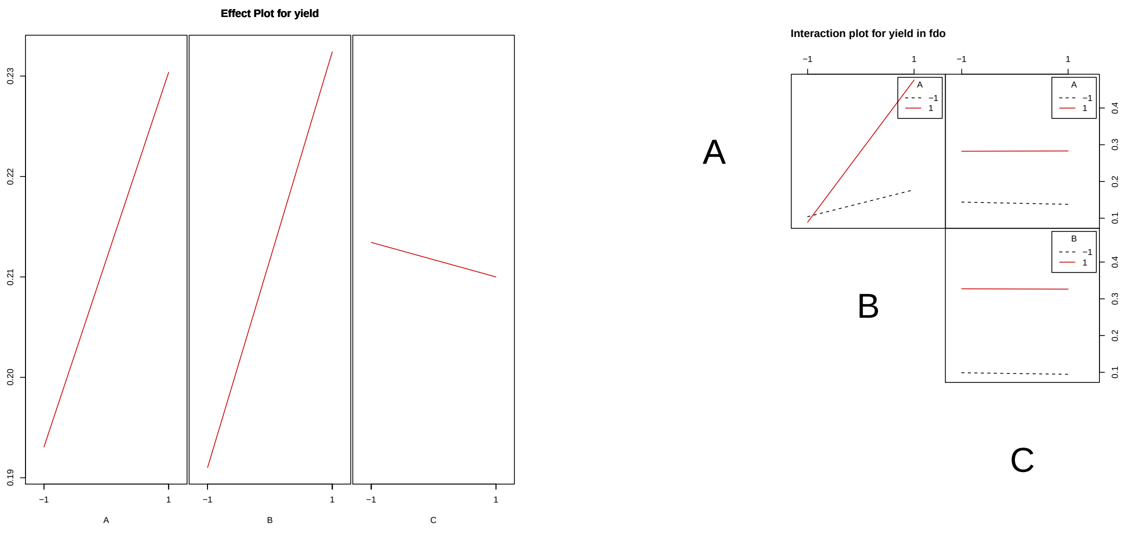


Figure 7: effect- and interaction plot for the factorial design

```
Call:
lm(formula = yield ~ A * B * C, data = fdo)

Residuals:
...
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.2136547  0.0017484 122.198 2.69e-08 ***
A            0.0709026  0.0021414  33.111 4.96e-06 ***
B            0.1150478  0.0021414  53.726 7.18e-07 ***
C           -0.0012866  0.0021414  -0.601    0.580
A:B          0.0784133  0.0021414  36.618 3.32e-06 ***
A:C          0.0017168  0.0021414   0.802    0.468
B:C          0.0007944  0.0021414   0.371    0.729
A:B:C       -0.0014677  0.0021414  -0.685    0.531
---
Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’␣’ 1

Residual standard error: 0.006057 on 4 degrees of freedom
Multiple R-squared: 0.9992,     Adjusted R-squared: 0.9979
F-statistic: 760.8 on 7 and 4 DF,  p-value: 4.431e-06
```

The effects of A and B as well as the interaction A:B are identified to be significant. A Pareto plot of the standardized effects visualizes these findings and can be created with the paretoPlot method of the qualityTools package (figure 8).

```
> paretoPlot(fdo)
```

The relation between the factors A and B can be visualized as 3D representation in form of a wireframe or contour plot using the wirePlot and contourPlot method of the qualityTools package (figure 11). Again, no further transformation of the data is needed!

```
> wirePlot(A, B, yield, data = fdo)
> contourPlot(A, B, yield, data = fdo)
```

One question that arises is whether this linear fit adequately describes the process. In order to find out, one can simply compare values predicted in the center of the design (i.e.

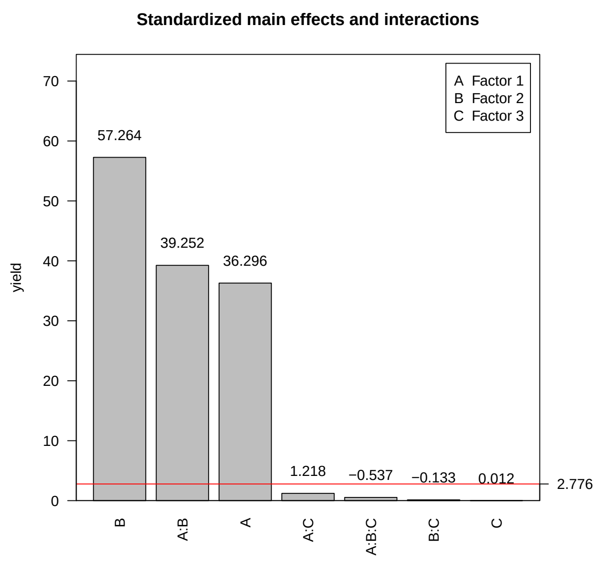


Figure 8: pareto plot of the standardized effects

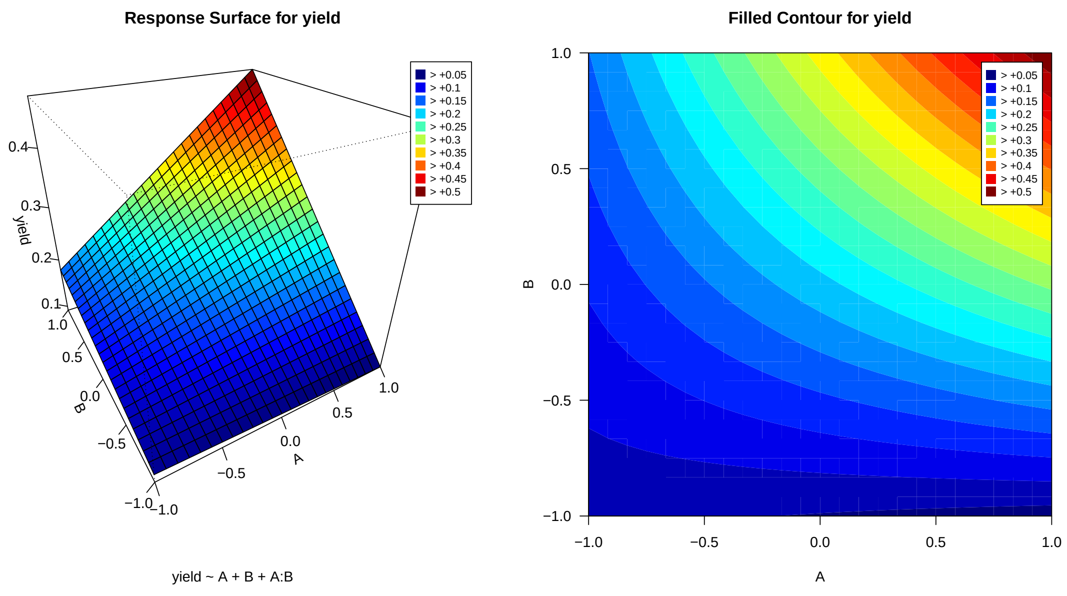


Figure 9: response surface and contour plot

A=0, B=0 and C=0) with the values observed in the center of the design. This difference could also be tested using a specialized t-Test. For now, let's assume the model is less wrong than others (i.e. we don't know of any better model).

5.2. 2^{k-p} Fractional Factorial Designs

Imagine testing 5 different factors in a 2^k design giving you $2^5 = 32$ runs. This is likely to be quite expensive if run on any machine, process or setting within production, research or a similar environment. Before dismissing the design, it's advisable to reflect what this design is capable of in terms of what types of interactions it can estimate. The highest interaction in a 2^5 design is the interaction between the five factors ABCDE. This interaction, even if significant, is really hard to interpret, and likely to be non-existent. The same applies for interactions between four factors and some of the interactions between 3 factors which is why most of the time fractional factorial designs are considered in the first stages of experimentation.

A fractional factorial design is denoted 2^{k-p} meaning k factors are tested in 2^{k-p} runs. In a 2^{5-1} design five factors are tested in 24 runs (hence p=1 additional factor is tested without further runs). This works by confounding interactions with additional factors. This section will elaborate on this idea with the help of the methods of the qualityTools package.

For fractional factorial designs the method `fracDesign` of the qualityTools package can be used. The generators can be given in the same notation that is used in textbooks on this matter. For a 2^{3-1} design (i.e. 3 factors that are to be tested in a 2^2 by confounding the third factor with the interaction between the first two factors) this would be given by the argument gen = "C = AB" meaning the interaction between A and B is to be confounded with the effect of a third factor C. The effect estimated for C is then confounded with the interaction AB; they cannot be separately estimated, hence C = AB (alias) or the alias of C is AB.

```
> fdo.frac = fracDesign(k = 3, gen = "C␣=␣AB", centerCube = 4)
```

In order to get more specific information about a design the `summary` method can be used. For this example you will see on the last part the identity I = ABC of the design. The identity I of a design is the left part of the generator multiplied by the generator. The resolution is the (character-) length of the shortest identity.

```
> summary(fdo.frac)

Information about the factors:

            A       B       C
low        -1      -1      -1
high        1       1       1
name
unit
type numeric numeric numeric

-----------
  StandOrd RunOrder Block  A  B  C  y
1        1        1     1 -1 -1  1 NA
2        2        2     1  1 -1 -1 NA
3        3        3     1 -1  1 -1 NA
4        4        4     1  1  1  1 NA
5        5        5     1  0  0  0 NA
```

```
6        6        6       1  0  0  0 NA
7        7        7       1  0  0  0 NA
8        8        8       1  0  0  0 NA

---------

Defining relations:
I =  ABC                Columns: 1 2 3

Resolution:  III
```

The following rules apply

$$I \times A = A \tag{5}$$
$$A \times A = I \tag{6}$$
$$A \times B = B \times A \tag{7}$$

By multiplying A, B and C you will find all confounded effects or aliases. A more convenient way to get an overview of the alias structure of a factorial design is to call the method `aliasTable` or `confounds` of the qualityTools package. The latter gives a more human readable version of the first.

```
> aliasTable(fdo.frac)

         C  AC  BC  ABC
Identity 0  0   0    1
A        0  0   1    0
B        0  1   0    0
AB       1  0   0    0
```

Fractional factorial designs can be generated by assigning the appropriate generators. However, most of the time standard fractional factorial designs known as minimum aberration designs [4] will be used. Such a design can be chosen from predefined tables by using the method `fracChoose` of the qualityTools package and simply clicking onto the desired design (figure 10).

```
> fracChoose()
```

5.3. Replicated Designs and Center Points

A replicated design with additional center points can be created by using the `replicates` and `centerCube` argument.

```
> fdo1 = facDesign(k = 3, centerCube = 2, replicates = 2)
```

5.4. Multiple Responses

Once you have observed the response for the different factor combinations one can add one or more response vectors to the design with the `response` method of the qualityTools package . A second response to be named y2 is created with the help of random numbers.

```
> set.seed(1234)
> y2 = rnorm(12, mean = 20)
> response(fdo) = data.frame(yield, y2)
```

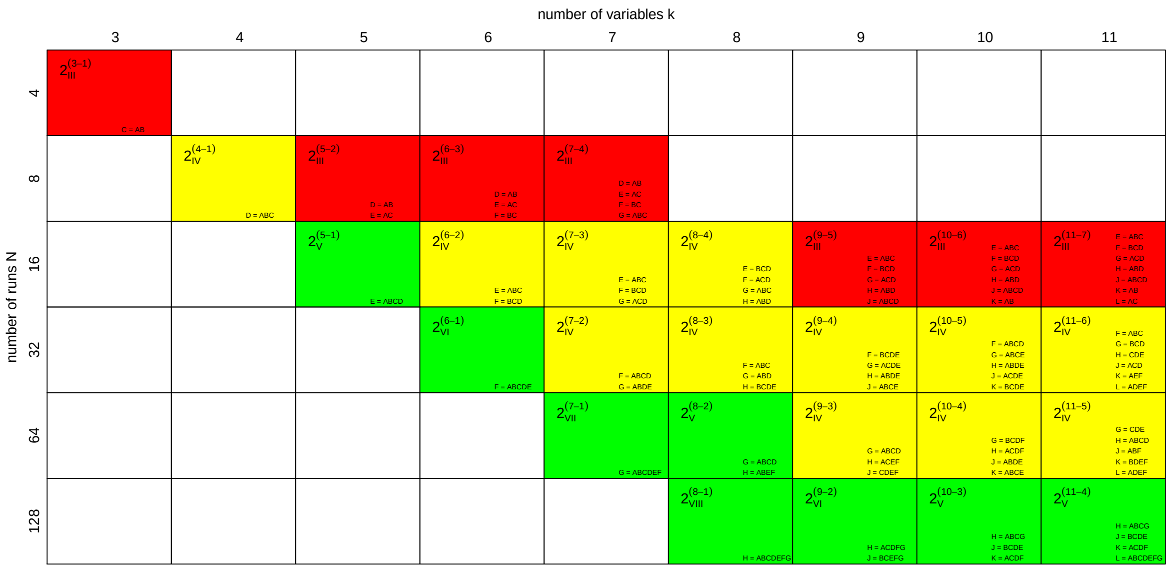


Figure 10: Choosing minimum aberration designs

A 3D visualization is done with the help of the methods wirePlot and contourPlot of the qualityTools package with no need to first create arrays of values or the like. Simply specify the formula you would like to fit with e.g. form = "yield ∼ A+B". Specifying this fit for response yield one can see that there's actually no practical difference to the fit that included an interaction term (figure 11).

```
> par(mfrow = c(1,2))
> wirePlot(A, B, yield, data = fdo, form = "yield~A+B+C+A*B")
> contourPlot(A, B, y2, data = fdo, form = "y2~A+B+C+A*B")
```

Using the wirePlot and contourPlot methods of the qualityTools package settings of the other n-2 factors can be set using the factors argument. A wireplot with the third factor C on -1 an C = 1 can be created as follows (figure 12)

```
> wirePlot(A,B,y2, data = fdo, factors = list(C=-1), form = "y2~A*B*C")
> wirePlot(A,B,y2, data = fdo, factors = list(C=1), form = "y2~A*B*C")
```

If no formula is explicitly given the methods default to the full fit or the fit stored in the factorial design object fdo. Storing a fit can be done using the fits method of the qualityTools package and is especially useful when working with more than one response (see 5.4). Of course lm can be used to analyze the fractional factorial designs.

```
> fits(fdo) = lm(yield ~ A+B, data = fdo)
> fits(fdo) = lm(y2 ~ A*B*C, data = fdo)
> fits(fdo)

$yield

Call:
lm(formula = yield ~ A + B, data = fdo)

Coefficients:
...
```

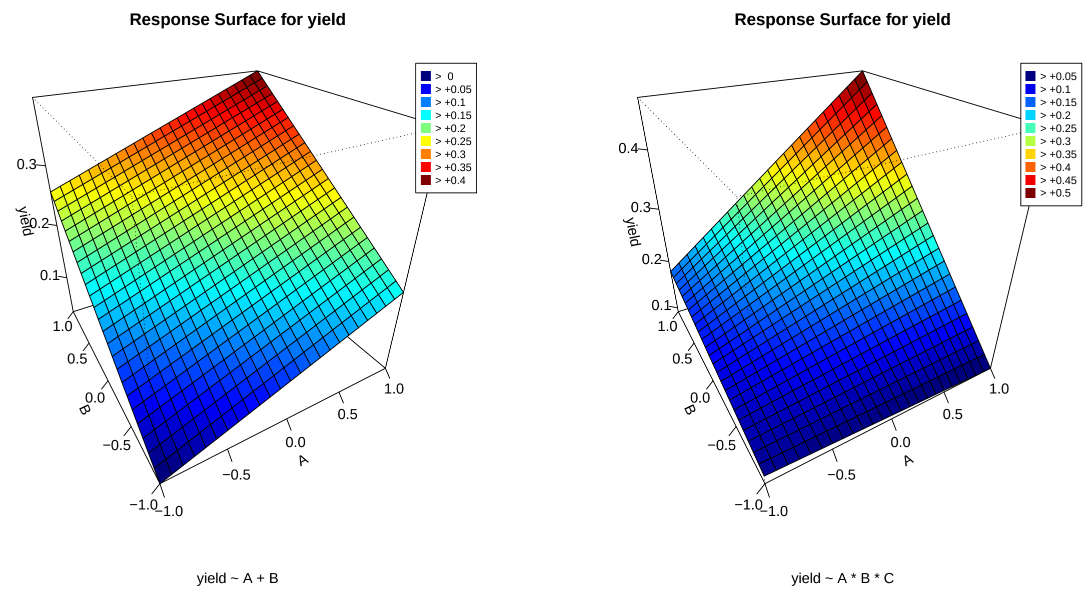


Figure 11: wire plot with different formulas specified

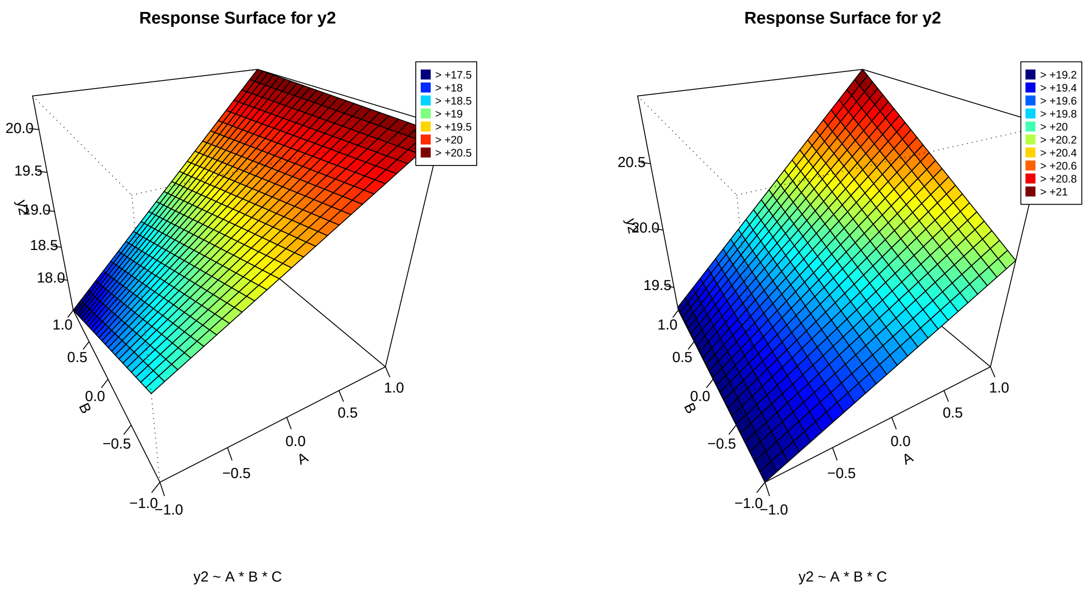


Figure 12: wire plot with formula and setting for factor C

5.5. Moving to a process setting with an expected higher yield

Since our process can be adequately modeled by a linear relationship the direction in which to go for an expected higher yield is easy to determine. A contour plot of factor A and B illustrate that we simply need to "step up the stairs". The shortest way to get up these stairs (figure 9) can be figured out graphically or calculated using the `steepAscent` method of the qualityTools package.

```
> sao =steepAscent(factors=c("A","B"),response="yield",data=fdo,steps
    =20)
> sao

Steepest Ascent for fdo

   Run Delta A.coded B.coded A.real B.real
1    1     0     0.0   0.000    100    130
...
12  12    11     2.2   3.570    144    166
13  13    12     2.4   3.894    148    169
14  14    13     2.6   4.219    152    172
...
21  21    20     4.0   6.490    180    195
```

Since we set the real values earlier using the `highs` and `lows` methods of the qualityTools package factors settings are displayed in coded as well as real values. Again the values of the response of sao[7] can be set using the `response` method of the qualityTools package and then be plotted using the `plot` method. Of course one can easily use the base plot method itself. However for documentation purposes the `plot` method for a steepest ascent object might be more convenient.

```
> predicted = simProc(sao[,5], sao[,6])
> response(sao) = predicted
> plot(sao, type = "b")
```

At this point the step size was chosen quite small for illustration purposes.

5.6. Response Surface Designs

Not all relations are linear and thus in order to detect and model non-linear relationships sometimes more than two combinations per factor are needed. At the beginning all a black box might need is a 2^k or 2^{k-p} design. In order to find out whether a response surface design (i.e. a design with more than two combination per factors) is needed one can compare the expected value of one's response variable(s) with the observed one(s) using centerpoints (i.e. the 0, 0, ..., 0 setting). The bigger the difference between observed and expected values, the more unlikely this difference is the result of random noise.

For now, let's return to the initial simulated process. The project started off with a 2^k design containing center points. Sticking to a linear model we used the `steepAscent` method of the qualityTools package to move to a better process region. The center of the new process region is defined by 144 and 165 in real values. This region is the start of a new design. Again one starts by using a factorial design

```
#set the seed for randomization of the runs
> set.seed(1234)
> fdo2 = facDesign(k = 2, centerCube = 3)
```

[7]steepest ascent object

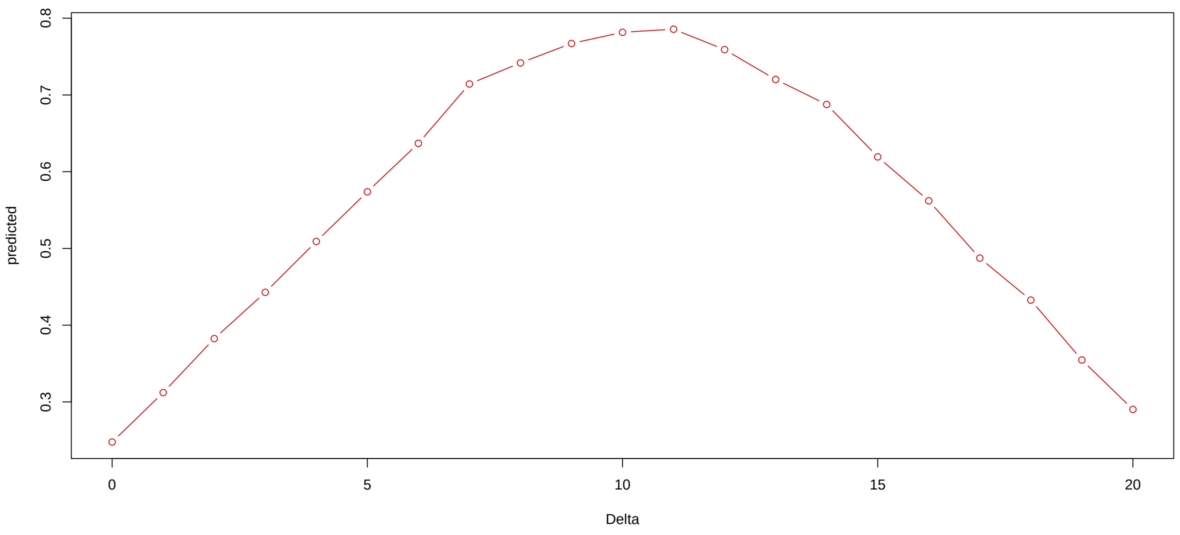


Figure 13: predicted maximum at Delta = 11 (see sao)

```
> names(fdo2) = c("Factor␣1", "Factor␣2")
> lows(fdo2) = c(134, 155)
> highs(fdo2) = c(155, 175)
```

and the yield is calculated by using the simProc and assigned to the design with the help of the generic response method of the qualityTools package.

```
> yield = c(simProc(134,155), simProc(155,155), simProc(134,175),
   simProc(155,175), simProc(144,165), simProc(144,165), simProc
   (144,165))
> response(fdo2) = yield
```

Looking at the residual graphics one will notice a substantial difference between expected and observed values (a test for lack of fit could of course be performed and will be significant). To come up with a model that describes the relationship one needs to add further points which are referred to as the star portion of the response surface design.

Adding the star portion is easily done using the starDesign method of the qualityTools package. By default the value of alpha is chosen so that both criteria, **orthogonality** and **rotatability** are approximately met. Simply call the starDesign method on the factorial design object fdo2. Calling rsdo[8] will show you the resulting response surface design. It should have a cube portion consisting of 4 runs, 3 center points in the cube portion, 4 axial and 3 center points in the star portion.

```
> rsdo = starDesign(data = fdo2)
```

Using the star method of the qualityTools package one can easily assemble designs sequentially. This sequential strategy saves resources since compared to starting off with a response surface design from the very beginning, the star portion is only run if really needed. The yields for the process are still given by the simProc method of the qualityTools package.

[8]response surface design object

```
> yield2 = c(yield, simProc(130,165), simProc(149,165), simProc(144,151)
    ,simProc(144,179),simProc(144,165),simProc(144,165),simProc(144,165)
  )
> response(rsdo) = yield2
```

A full quadratic model is fitted using the `lm` method

```
> lm.3 = lm(yield2 ~ A*B + I(A^2) + I(B^2), data = rsdo)
```

and one sees that there are significant quadratic components. The response surface can be visualized using the `wirePlot` and `contourPlot` method of the qualityTools package.

```
> wirePlot(A,B,yield2,form="yield2~A*B+I(A^2)+I(B^2)",data=rsdo,theta
  =-70)
> contourPlot(A,B,yield2,form="yield2~A*B+I(A^2)+I(B^2)",data=rsdo)
```

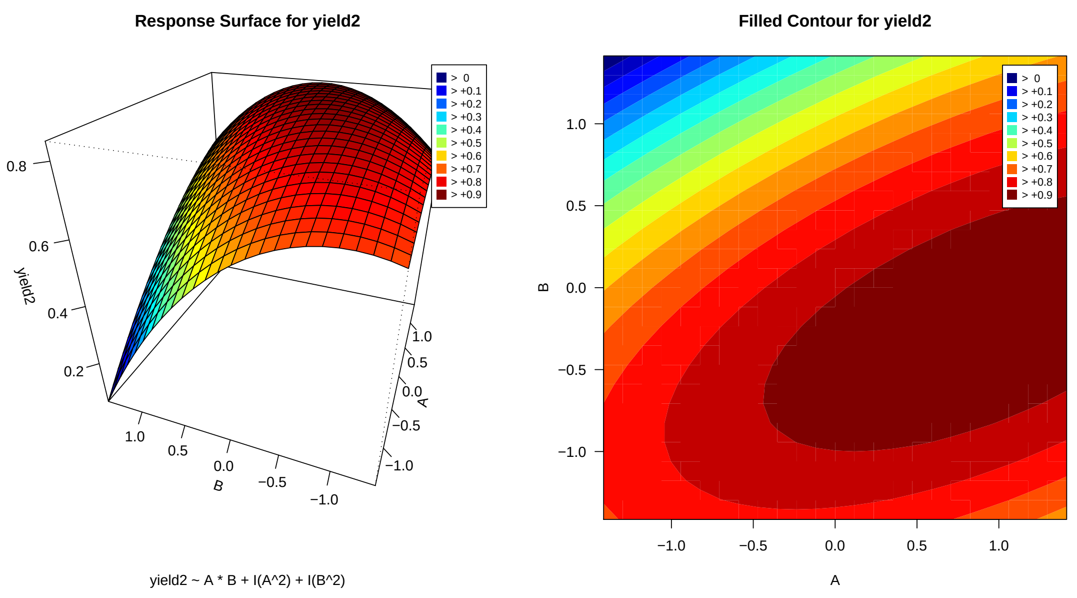


Figure 14: quadratic fit of the response surface design object rsdo

Figure 15 can be used to compare the outcomes of the factorial and response surface designs with the simulated process. The inactive Factor 3 was omitted.

Besides this sequential strategy, response surface designs can be created using the method `rsmDesign` of the qualityTools package. A design with `alpha = 1.633`, 0 center-points in the cube portion and 6 center points in the star portion can be created with:

```
> fdo = rsmDesign(k = 3, alpha = 1.633, cc = 0, cs = 6)
```

and the design can be put in standard order using the randomize method with argument `so=TRUE` (i.e. standard order). `cc` stands for centerCube and `cs` for centerStar.

```
> fdo = randomize(fdo, so = TRUE)
```

Response Surface Designs can also be chosen from a table by using the method `rsmChoose` of the qualityTools package.

```
> rsdo = rsmChoose()
```

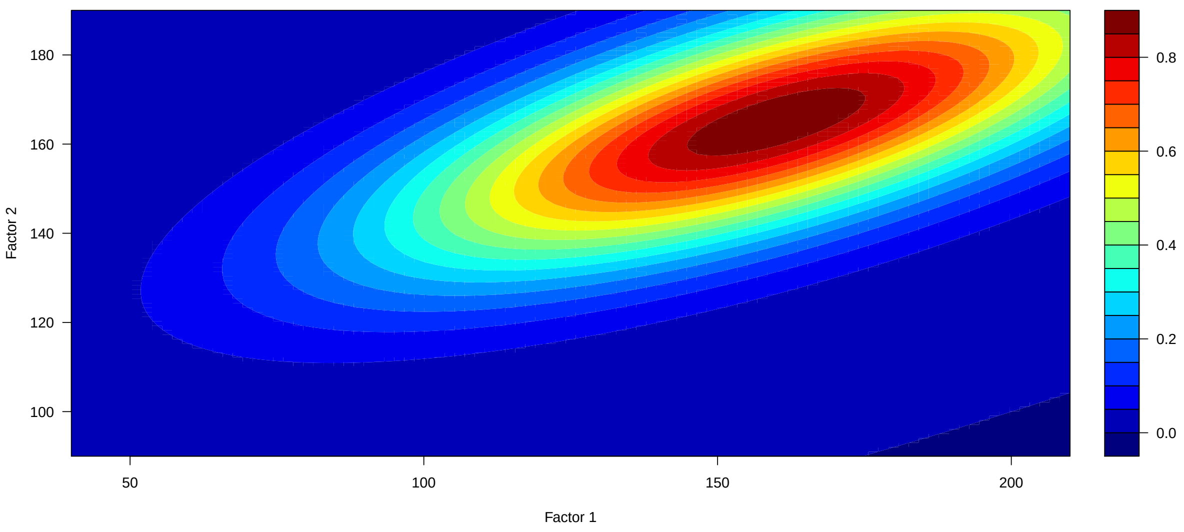
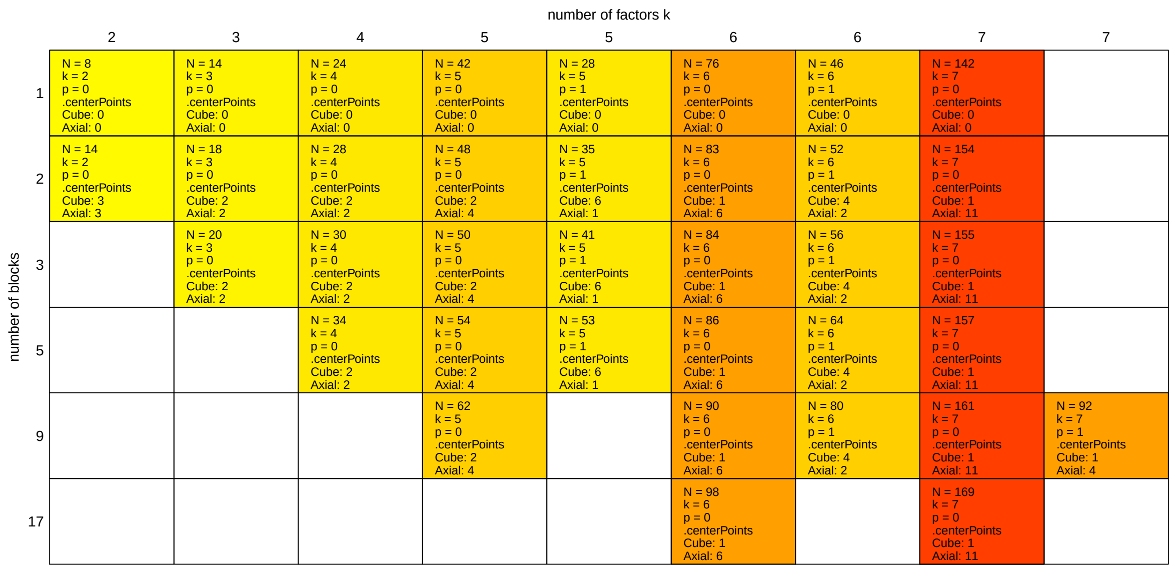


Figure 15: underlying black box process without noise



number of factors k

number of blocks	2	3	4	5	5	6	6	7	7
1	N = 8 k = 2 p = 0 .centerPoints Cube: 0 Axial: 0	N = 14 k = 3 p = 0 .centerPoints Cube: 0 Axial: 0	N = 24 k = 4 p = 0 .centerPoints Cube: 0 Axial: 0	N = 42 k = 5 p = 0 .centerPoints Cube: 0 Axial: 0	N = 28 k = 5 p = 1 .centerPoints Cube: 0 Axial: 0	N = 76 k = 6 p = 0 .centerPoints Cube: 0 Axial: 0	N = 46 k = 6 p = 1 .centerPoints Cube: 0 Axial: 0	N = 142 k = 7 p = 0 .centerPoints Cube: 0 Axial: 0	
2	N = 14 k = 2 p = 0 .centerPoints Cube: 3 Axial: 3	N = 18 k = 3 p = 0 .centerPoints Cube: 2 Axial: 2	N = 28 k = 4 p = 0 .centerPoints Cube: 2 Axial: 2	N = 48 k = 5 p = 0 .centerPoints Cube: 2 Axial: 4	N = 35 k = 5 p = 1 .centerPoints Cube: 6 Axial: 1	N = 83 k = 6 p = 0 .centerPoints Cube: 1 Axial: 6	N = 52 k = 6 p = 1 .centerPoints Cube: 4 Axial: 2	N = 154 k = 7 p = 0 .centerPoints Cube: 1 Axial: 11	
3		N = 20 k = 3 p = 0 .centerPoints Cube: 2 Axial: 2	N = 30 k = 4 p = 0 .centerPoints Cube: 2 Axial: 2	N = 50 k = 5 p = 0 .centerPoints Cube: 2 Axial: 4	N = 41 k = 5 p = 1 .centerPoints Cube: 6 Axial: 1	N = 84 k = 6 p = 0 .centerPoints Cube: 1 Axial: 6	N = 56 k = 6 p = 1 .centerPoints Cube: 4 Axial: 2	N = 155 k = 7 p = 0 .centerPoints Cube: 1 Axial: 11	
5			N = 34 k = 4 p = 0 .centerPoints Cube: 2 Axial: 2	N = 54 k = 5 p = 0 .centerPoints Cube: 2 Axial: 4	N = 53 k = 5 p = 1 .centerPoints Cube: 6 Axial: 1	N = 86 k = 6 p = 0 .centerPoints Cube: 1 Axial: 6	N = 64 k = 6 p = 1 .centerPoints Cube: 4 Axial: 2	N = 157 k = 7 p = 0 .centerPoints Cube: 1 Axial: 11	
9				N = 62 k = 5 p = 0 .centerPoints Cube: 2 Axial: 4		N = 90 k = 6 p = 0 .centerPoints Cube: 1 Axial: 6	N = 80 k = 6 p = 1 .centerPoints Cube: 4 Axial: 2	N = 161 k = 7 p = 0 .centerPoints Cube: 1 Axial: 11	N = 92 k = 7 p = 1 .centerPoints Cube: 1 Axial: 4
17						N = 98 k = 6 p = 0 .centerPoints Cube: 1 Axial: 6		N = 169 k = 7 p = 0 .centerPoints Cube: 1 Axial: 11	

Figure 16: choosing a predefined response surface design from a table

5.6.1. Sequential Assembly of Response Surface Designs

Sequential assembly is a very important feature of Response Surface Designs. Depending on the features of the (fractional) factorial design a star portion can be augmented using the `starDesign` method of the qualityTools package. A star portion consists of axial runs and optional center points (`cs`) in the axial part as opposed to center points (`cc`) in the cube part.

```
> fdo3 = facDesign(k = 6)
> rsdo = starDesign(alpha = "orhtogonal", data = fdo3)
```

In case no existing (fractional) factorial design is handed to the `starDesign` method a list with `data.frames` is returned which can be assigned to the existing (fractional) factorial design using the `star`, `centerStar` and `centerCube` methods of the qualityTools package.

5.6.2. Randomization

Randomization is achieved by using the `randomize` method of the qualityTools package. At this point randomization works for most of the designs types. A `random.seed` needs to be supplied which is helpful to have the same run order on any machine.

```
> randomize(fdo, random.seed = 123)
```

5.6.3. Blocking

Blocking is another relevant feature and can be achieved by the `blocking` method of the qualityTools package. At this point blocking a design afterwards is not always successful. However, it is unproblematic during the sequential assembly.

5.7. Desirabilites

Many problems involve the simultaneous optimization of more than one response variable. Optimization can be achieved by either maximizing or minimizing the value of the response or by trying to set the response on a specific target. Optimization using the Desirabilities approach [6], the (predicted) values of the response variables are transformed into values within the interval [0,1] using three different desirability methods for the three different optimization criterias (i.e. minimize, maximize, target). Each value of a response variable can be assigned a specific desirability, optimizing more than one response variable. The geometric mean of the specific desirabilities characterizes the overall desirability.

$$\sqrt[n]{\prod_{i=1}^{n} d_i} \tag{8}$$

This means, for the predicted values of the responses, each factor combination has a corresponding specific desirability and an overall desirability can be calculated. Suppose we have three responses. For a specific setting of the factors the responses have desirabilities such as $d_1 = 0.7$ for y_1, $d_2 = 0.8$ for y_2 and $d_3 = 0.2$ for y_3. The overall desirability d_{all} is then given by the geometric mean

$$d_{all} = \sqrt[n]{d_1, d_2, \ldots, d_n} \tag{9}$$

$$= \sqrt[3]{d_1, d_2, d_3} \tag{10}$$

$$= \sqrt[3]{0.7 \cdot 0.8 \cdot 0.2} \tag{11}$$

Desirability methods can be defined using the `desires` method of the qualityTools package. The optimization direction for each response variable is defined via the `min`, `max` and `target` argument of the `desires` method. The `target` argument is set with `max` for maximization, `min` for minimization and a specific value for optimization towards a specific target. Three settings arise from this constellation

target = max: min is the lowest acceptable value. If the response variable takes values below min the corresponding desirability will be zero. For values equal or greater than min the desirability will be greater zero.

target = min: max is the highest acceptable value. If the response variable takes values above max the corresponding desirability will be zero. For values equal or less than max the desirability will be greater zero.

target = value: a response variable with a value of value relates to the highest achievable desirability of 1. Values outside min or max lead to a desirability of zero, inside min and max to values within (0,1]

Besides these settings the scale factor influences the shape of the `desirability` method. Desirability methods can be created and plotted using the `desires` and `plot` method of the qualityTools package. Desirabilities are always attached to a response and thus should be assigned to factorial designs (figure 17).

```
> d1 = desirability(y1, 120, 170, scale = c(1,1), target = "max")
> d3 = desirability(y3, 400, 600, target = 500)
> d1

Target is to maximize y1
lower Bound:  120
higher Bound:  170
Scale factor is:  1 1
importance:  1
```

Besides having a summary on the command line, the `desirability` method can be conveniently visualized using the `plot` method. With the desirabilities d1 and d3 one gets the following plots.

```
> plot(d1); plot(d3)
```

5.8. Using desirabilities together with designed experiments

The `desirability` methodology is supported by the factorial design objects. The output of the `desirability` method can be stored in the design object, so that information that belongs to each other is stored in the same place (i.e. the design itself). In the following

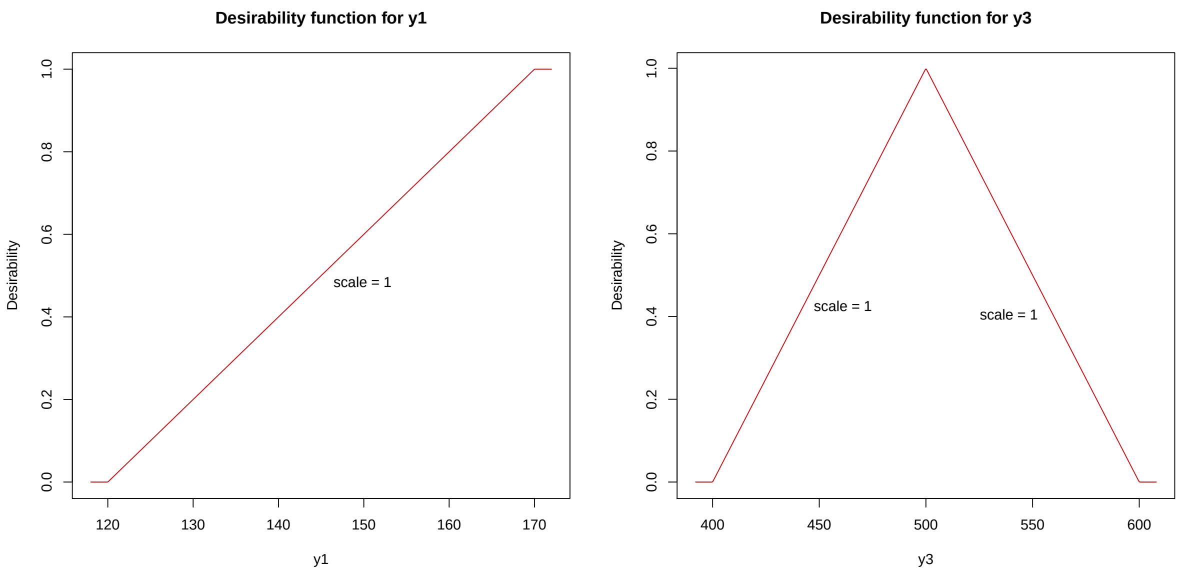


Figure 17: plotted desirabilities for y1 and y3

few R lines a designed experiment that uses desirabilities will be shown. The data used comes from [6]. Four responses y1, y2, y3, and y4 were defined. Factors used in this experiment were silica, silan, and sulfur with high factor settings of 1.7, 60, 2.8 and low factor settings of 0.7, 40, 1.8. It was desired to have y1 and y2 maximized and y3 and y4 set on a specific target (see below).

First of all the corresponding design that was used in the paper is created using the method `rsmDesign` of the qualityTools package. Then we use the `randomize` method to obtain the standard order of the design.

```
> ddo = rsmDesign(k = 3, alpha = 1.633, cc = 0, cs = 6)
> ddo = randomize(ddo, so = TRUE)
#optional
> names(ddo) = c("silica", "silan", "sulfur")
#optional
> highs(ddo) = c(1.7, 60, 2.8)
#optional
> lows(ddo) = c(0.7, 40, 1.8)
```

The `summary` method gives an overview of the design. The values of the responses are incorporated with the `response` method of the qualityTools package.

```
> y1 = c(102, 120, 117, 198, 103, 132, 132, 139, 102, 154, 96, 163, 116,
    153, 133, 133, 140, 142, 145, 142)
> y2 = c(900, 860, 800, 2294, 490, 1289, 1270, 1090, 770, 1690, 700,
   1540, 2184, 1784, 1300, 1300, 1145, 1090, 1260, 1344)
> y3 = c(470, 410, 570, 240, 640, 270, 410, 380, 590, 260, 520, 380,
   520, 290, 380, 380, 430, 430, 390, 390)
> y4 = c(67.5, 65, 77.5, 74.5, 62.5, 67, 78, 70, 76, 70, 63, 75, 65, 71,
    70, 68.5, 68, 68, 69, 70)
```

The sorted `data.frame` of these 4 responses is assigned to the design object ddo.

```
> response(ddo) = data.frame(y1, y2, y3, y4)[c(5,2,3,8,1,6,7,4,9:20),]
```

The desirabilities are incorporated with the desires method of the qualityTools package. y1 and y3 were already defined which leaves the desirabailities for y2 and y4 to be defined.

```
> d2 = desirability(y2, 1000, 1300, target = "max")
> d4 = desirability(y4, 60, 75, target = 67.5)
```

The desirabilities need to be defined with the names of the response variables in order to use them with the responses of the design object. The desires method is used as follows.

```
> desires(ddo)=d1; desires(ddo)=d2; desires(ddo)=d3; desires(ddo)=d4
```

Fits are set as in [6] using the fits methods of the qualityTools package.

```
> fits(ddo) = lm(y1 ~ A+B+C+A:B+A:C+B:C+I(A^2)+I(B^2)+I(C^2), data = ddo
  )
> fits(ddo) = lm(y2 ~ A+B+C+A:B+A:C+B:C+I(A^2)+I(B^2)+I(C^2), data = ddo
  )
> fits(ddo) = lm(y3 ~ A+B+C+A:B+A:C+B:C+I(A^2)+I(B^2)+I(C^2), data = ddo
  )
> fits(ddo) = lm(y4 ~ A+B+C+A:B+A:C+B:C+I(A^2)+I(B^2)+I(C^2), data = ddo
  )
```

The overall optimum can now be calculated using the method optimum of the qualityTools package giving the same factor settings as stated in [6] for an overall desirability of 0.58 and individual desirabilities of 0.187, 1, 0.664, 0.934 for y1, y2, y3 and y4.

```
> optimum(ddo, type = "optim")

composite (overall) desirability: 0.583

            A     B      C
coded -0.0533 0.144 -0.872
real  -0.0533 0.144 -0.872

                     y1   y2      y3     y4
Responses       129.333 1300 466.397 67.997
Desirabilities    0.187    1   0.664  0.934
```

5.9. Mixture Designs

At this time the generation of the different kinds of mixture designs is fully supported including a ternary contour and 3D plot. Analyzing these designs however needs to be done without any specific support by a method of the qualityTools package.

Following the introduced name convention of the qualityTools package the method mixDesign can be used to e.g. create simplex lattice design and simplex centroid designs. The generic methods response, names, highs, lows, units and types are again supported. A famous data set [5] is given by the elongation of yarn for various mixtures of three factors. This example can be reconstructed using the method mixDesign of the qualityTools package. mdo is an abbreviation of mix design object.

```
> mdo = mixDesign(3,2, center = FALSE, axial = FALSE, randomize = FALSE,
    replicates  = c(1,1,2,3))
> names(mdo) = c("polyethylene", "polystyrene", "polypropylene")

#set response (i.e. yarn elongation)
```

```
> elongation = c(11.0, 12.4, 15.0, 14.8, 16.1, 17.7, 16.4, 16.6, 8.8,
    10.0, 10.0, 9.7, 11.8, 16.8, 16.0)
> response(mdo) = elongation
```

Again the values of the response are associated with the method `response` of the qualityTools package. Calling mdo prints the design. The generic `summary` method can be used for a more detailed overview.

```
> mdo
```

```
   StandOrder RunOrder    Type   A   B   C elongation
1           1        1 1-blend 1.0 0.0 0.0       11.0
2           2        2 1-blend 1.0 0.0 0.0       12.4
3           3        3 2-blend 0.5 0.5 0.0       15.0
4           4        4 2-blend 0.5 0.5 0.0       14.8
5           5        5 2-blend 0.5 0.5 0.0       16.1
6           6        6 2-blend 0.5 0.0 0.5       17.7
7           7        7 2-blend 0.5 0.0 0.5       16.4
8           8        8 2-blend 0.5 0.0 0.5       16.6
9           9        9 1-blend 0.0 1.0 0.0        8.8
10         10       10 1-blend 0.0 1.0 0.0       10.0
11         11       11 2-blend 0.0 0.5 0.5       10.0
12         12       12 2-blend 0.0 0.5 0.5        9.7
13         13       13 2-blend 0.0 0.5 0.5       11.8
14         14       14 1-blend 0.0 0.0 1.0       16.8
15         15       15 1-blend 0.0 0.0 1.0       16.0
```

The data can be visualized using the `wirePlot3` and `contourPlot3` methods (figure 18). In addition to the `wirePlot` and `contourPlot` methods the name of the third factor (i.e. C) and the type of standard fit must be given. Of course it is possible to specify a fit manually using the `form` argument with a formula.

```
> contourPlot3(A, B, C, elongation, data = mdo, form = "quadratic")
> wirePlot3(A, B, C, elongation, data=mdo, form="quadratic", theta=-170)
```

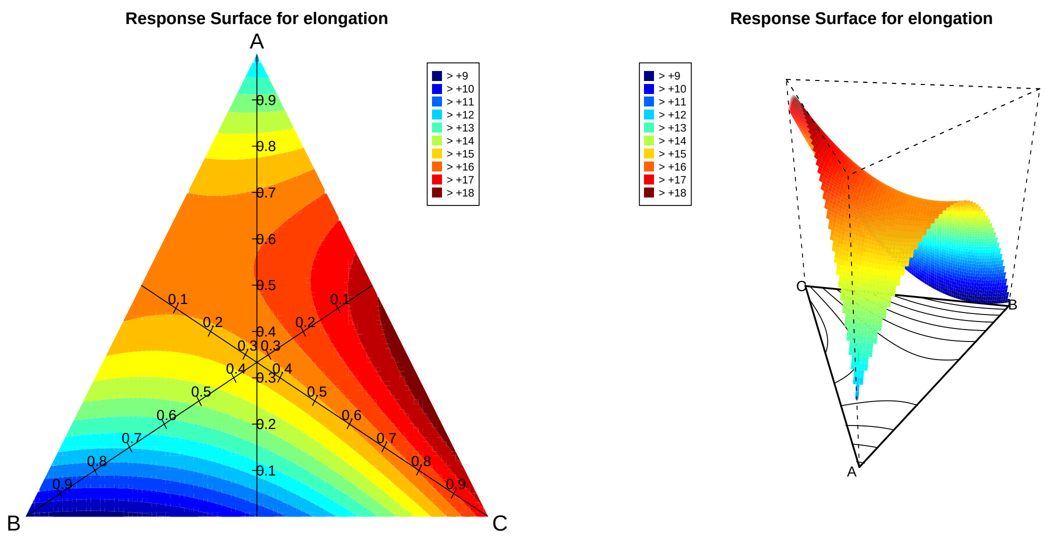


Figure 18: ternary plots for the elongation example

5.10. Taguchi Designs

Taguchi Designs are available using the method taguchiDesign of the qualityTools package. There are two types of taguchi designs:

- Single level: all factors have the same number of levels (e.g. two levels for a L4_2)
- Mixed level: factors have different number of levels (e.g. two and three levels for L18_2_3)

Most of the designs that became popular as taguchi designs however are simple 2^k fractional factorial designs with a very low resolution of III (i.e. main effects are confounded with two factor interactions) or other mixed level designs and are originally due to contributions by other e.g. Plackett and Burman, Fisher, Finney and Rao [3]. A design can be created using the taguchiDesign method of the qualityTools package. The generic method names, units, values, summary, plot, lm and other methods again are supported. This way the relevant information for each factor can be stored in the design object tdo[9] itself.

```
> set.seed(1234)
> tdo = taguchiDesign("L9_3")
> values(tdo) = list(A  = c(20, 40, 60), B = c("mateial␣1", "material␣2"
  , "material␣3"), C = c(1,2,3))
> names(tdo) = c("Factor␣1", "Factor␣2", "Factor␣3", "Factor␣4")
> summary(tdo)

Taguchi SINGLE Design
Information about the factors:

                A          B        C        D
value 1        20   mateial 1        1        1
value 2        40  material 2        2        2
value 3        60  material 3        3        3
name     Factor 1   Factor 2 Factor 3 Factor 4
unit
type      numeric    numeric  numeric  numeric

-----------

  StandOrder RunOrder Replicate A B C D  y
1          7        1         1 3 1 3 2 NA
2          1        2         1 1 1 1 1 NA
3          6        3         1 2 3 1 2 NA
4          4        4         1 2 1 2 3 NA
5          2        5         1 1 2 2 2 NA
6          8        6         1 3 2 1 3 NA
7          5        7         1 2 2 3 1 NA
8          3        8         1 1 3 3 3 NA
9          9        9         1 3 3 2 1 NA
```

The response method is used to assign the values of the response variables. effectPlot can be used once more to visualize the effect sizes of the factors (figure 19).

```
> response(tdo) = rnorm(9)
> effectPlot(tdo)
```

[9]taguchi design object

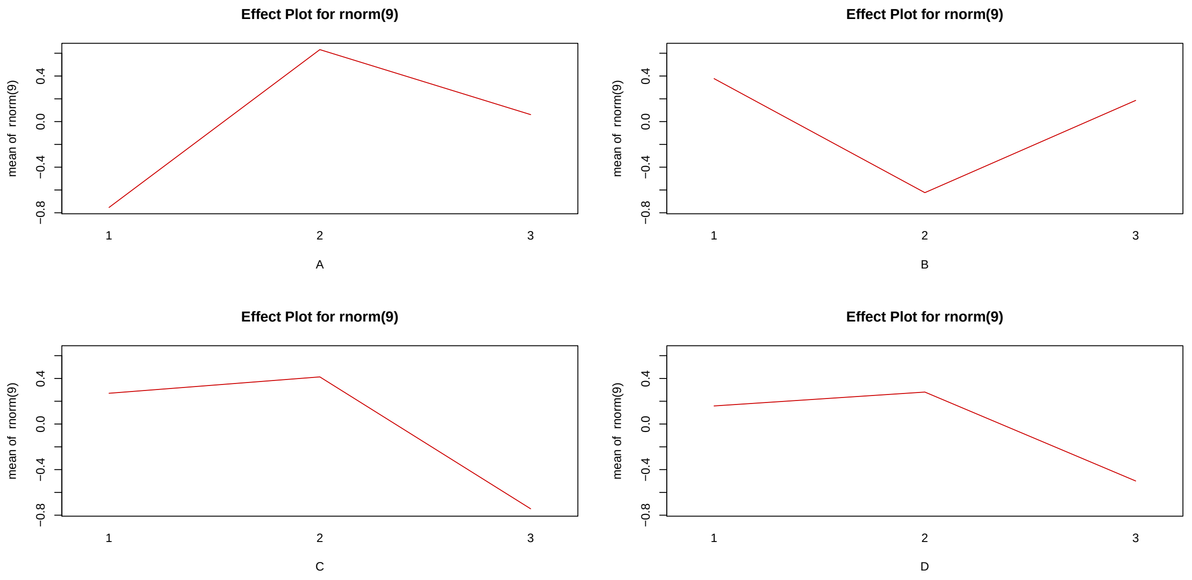


Figure 19: effect plot for the taguchi experiment

A. R-code

qualityTools in DEFINE

```
defects=c(rep("E",62),rep("B",15),rep("F",3),rep("A", 10),rep
("C",20),rep("D",10))
paretoChart(defects)
```

qualityTools in MEASURE

```
set.seed (1234)
design=gageRRDesign(Operators=3,Parts=9,Measurements=3)
response(design)=rnorm(3*9*3,mean=20)
gdo=gageRR(design)
plot(gdo)
```

qualityTools in ANALYZE

```
set.seed(1234)
norm=rnorm(20,mean=20)
weib=rweibull(20,shape=2,scale=8)
pcr(norm,"normal",lsl=17,usl=23)
pcr(weib,"weibull",usl=20)
par(mfrow=c(1 ,2))
qqPlot(weib,"weibull");qqPlot(weib,"normal")
ppPlot(norm,"weibull");ppPlot(norm,"normal")
```

qualityTools in IMPROVE

2^k Fractional Factorial Designs

```
set.seed(1234)
fdo=facDesign(k=3,centerCube=4)
names(fdo)=c("Factor␣1","Factor␣2","Factor␣3")
lows(fdo)=c(80,120,1)
highs(fdo)=c(120,140,2)
summary(fdo)
yield=simProc(x1=80,x2=120,x3=1)
yield=c(simProc(80,120,1),simProc(120,120,2),simProc(120,140,2),
simProc(80,140,1),simProc(120,120,1),simProc(120,140,1),simProc
(80,120,2),simProc(80,140,2),simProc(90,130,1.5),simProc(90,130,
1.5),simProc(90,130,1.5),simProc(90,130,1.5))
response(fdo)=yield
effectPlot(fdo)
interactionPlot(fdo)
lm.1=lm(yield~A*B*C,data=fdo)
summary(lm.1)
paretoPlot(fdo)
wirePlot(A,B,yield,data=fdo)
contourPlot(A,B,yield,data=fdo)
```

2^{k-p} Factorial Designs

```
fdo.frac=fracDesign(k=3,gen="C␣=␣AB",centerCube=4)
summary(fdo.frac)
aliasTable(fdo.frac)
fracChoose()
```

Replicated Designs and Center Points

```
fdo1=facDesign(k=3,centerCube=2,replicates=2)
```

Multiple Responses

```
set.seed(1234)
y2=rnorm(12,mean=20)
response(fdo)=data.frame(yield,y2)
par(mfrow=c(1 ,2))
wirePlot(A,B,yield, data=fdo,form="yield~A+B+C+A*B")
contourPlot(A,B, y2, data=fdo,form="y2~A+B+C+A*B")
wirePlot(A,B,y2,data=fdo,factors=list(C=-1),form="y2~A*B*C")
wirePlot (A,B,y2,data=fdo,factors=list(C=1),form="y2~A*B*C")
fits(fdo)=lm(yield~A+B,data=fdo)
fits(fdo)=lm(y2~A*B*C,data=fdo)
fits(fdo)
```

Moving to a process setting with an expected higher yield

```
sao=steepAscent(factors=c("A","B"),response="yield",data=fdo,steps=20)
sao
predicted=simProc(sao[,5],sao[,6])
response(sao)=predicted
plot(sao,type="b")
```

Response Surface Designs

```
set.seed(1234)
fdo2=facDesign(k=2,centerCube=3)
names(fdo2)=c("Factor␣1","Factor␣2")
lows(fdo2)=c(134,155)
highs(fdo2)=c(155,175)
yield=c(simProc(134,155),simProc(155,155),simProc(134,175),
simProc(155,175),simProc(144,165),simProc(144,165),simProc(144,165))
response(fdo2)=yield
rsdo=starDesign(data=fdo2)
yield2=c(yield,simProc(130,165),simProc(149,165),simProc(144,151),
    simProc(144,179)
,simProc(144,165),simProc(144,165),simProc(144,165))
response(rsdo)=yield2
lm.3=lm(yield2~A*B+I(A^2)+I(B^2),data=rsdo)
wirePlot(A,B,yield2,form="yield2~A*B+I(A^2)+I(B^2)",data =rsdo,theta
    =-70)
contourPlot(A,B,yield2,form="yield2~A*B+I(A^2)+I(B^2)",data=rsdo)
fdo=rsmDesign(k=3,alpha=1.633,cc=0,cs=6)
fdo=randomize(fdo,so=TRUE)
rsdo=rsmChoose()
```

Sequential Assembly of Response Surface Designs

```
fdo3=facDesign(k=6)
rsdo=starDesig(alpha="orhtogonal",data=fdo3)
```

Randomization

```
randomize(fdo,random.seed=123)
```

Desirabilites

```
d1=desirability(y1,120,170,scale=c(1,1),target="max")
d3=desirability(y3,400,600,target=500)
d1
plot(d1);plot(d3)
```

Using desirabilities together with designed experiments

```
ddo=rsmDesign(k=3,alpha=1.633,cc=0,cs=6)
ddo=randomize(ddo,so=TRUE)
names(ddo)=c("silica","silan","sulfur")
highs(ddo)=c(1.7,60,2.8)
lows(ddo)=c(0.7,40,1.8)
y1=c(102,120,117,198,103,132,132,139,102,154,96,163,116,
153,133,133,140,142,145,142)
y2=c(900,860,800,2294,490,1289,1270,1090,770,1690,700,1540,
2184,1784,1300,1300,1145,1090,1260,1344)
y3=c(470,410,570,240,640,270,410,380,590,260,520,380,520,
290,380,380,430,430,390,390)
y4=c(67.5,65,77.5,74.5,62.5,67,78,70,76,70,63,75,65,71,
70,68.5,68,68,69,70)
response(ddo)=data.frame(y1,y2,y3,y4)[c(5,2,3,8,1,6,7,4,9:20),]
d2=desirability(y2,1000,1300,target="max")
d4=desirability(y4,60,75,target=67.5)
desires(ddo)=d1; desires(ddo)=d2; desires(ddo)=d3; desires(ddo)=d4
fits(ddo)=lm(y1~A+B+C+A:B+A:C+B:C+I(A^2)+I(B^2)+I(C^2),data=ddo)
fits(ddo)=lm(y2~A+B+C+A:B+A:C+B:C+I(A^2)+I(B^2)+I(C^2),data=ddo)
fits(ddo)=lm(y3~A+B+C+A:B+A:C+B:C+I(A^2)+I(B^2)+I(C^2),data=ddo)
fits(ddo)=lm(y4~A+B+C+A:B+A:C+B:C+I(A^2)+I(B^2)+I(C^2),data=ddo)
optimum(ddo,type="optim")
```

Mixture Designs

```
mdo=mixDesign(3,2,center=FALSE,axial=FALSE,randomize=FALSE,
replicates=c(1,1,2,3))
names(mdo)=c("polyethylene","polystyrene","polypropylene")
elongation=c(11.0,12.4,15.0,14.8,16.1,17.7,16.4,16.6,8.8,10.0,
10.0,9.7,11.8,16.8,16.0)
response(mdo)=elongation
mdo
contourPlot3(A,B,C,elongation,data=mdo,form="quadratic")
wirePlot3(A,B,C,elongation,data=mdo,form="quadratic",theta=-170)
```

Taguchi Designs

```
set.seed(1234)
tdo=taguchiDesign("L9_3")
values(tdo)=list(A=c(20,40,60),B=c("mateial␣1","material␣2",
"material␣3"),C=c(1,2,3))
names(tdo)=c("Factor␣1","Factor␣2","Factor␣3","Factor␣4")
summary (tdo)
response(tdo)=rnorm(9)
effectPlot(tdo)
```

References

[1] *ISO 21747:2006. Statistical Methods - Process performance and capability statistics for measured quality characteristics.*

[2] *ISO 9000:2005. Quality Management Systems - Fundamentals and vocabulary.*

[3] Box, G.E.P., Bisgard S. and C. A. Fung: *An explanation and critique of taguchi's contributions to quality engineering.* Quality and Reliability Engineering International, 1988.

[4] Box, George E. P., Hunter J. Stuart and William G. Hunter: *Statistics for Experimenters.* Wiley, New Jersey, Second edition, 2005. ISBN 0-471-71813-0.

[5] Cornell, John: *Experiments with Mixtures: Designs, Models, and the Analysis of Mixture Data.* Wiley, New York, Third edition, 2002. ISBN-10: 0-471-39367-3.

[6] Derringer, George and Ronald Suich: *Simultaneous Optimization of Several Response Variables.* Journal of Quality Technology, 1980.

[7] Venables, W. N. and B. D. Ripley: *Modern Applied Statistics with S.* Springer, New York, Fourth edition, 2002. ISBN 0-387-95457-0.