Package: r6qualitytools (via r-universe)

February 1, 2025

Type Package

Title R6-Based Statistical Methods for Quality Science

Version 1.0.1.9000

Description A comprehensive suite of statistical tools for Quality Management, designed around the Define, Measure, Analyze, Improve, and Control (DMAIC) cycle used in Six Sigma methodology. Based on the discontinued CRAN package 'qualitytools', this package refactors its original design by incorporating 'R6' object-oriented programming for increased flexibility and performance. It replaces traditional graphics with modern, interactive visualizations using 'ggplot2' and 'plotly'. Built on 'tidyverse' principles, it simplifies data manipulation and visualization, offering an intuitive approach to quality science.

License GPL (>= 3)

URL https://github.com/Fabianenc/r6qualitytools

BugReports https://github.com/Fabianenc/r6qualitytools/issues

Imports dplyr, EnvStats, ggplot2, graphics, grid, gridExtra, MASS, methods, patchwork, plotly, R6, RColorBrewer, Rsolnp, scales, stats, tibble, tidyr, utils

Encoding UTF-8

RoxygenNote 7.3.1

Config/pak/sysreqs make libicu-dev libssl-dev

Repository https://fabianenc.r-universe.dev

RemoteUrl https://github.com/fabianenc/r6qualitytools

RemoteRef HEAD

RemoteSha 2f61dfd4e47606d0a0d0c711afc5b03701906f23

Contents

adSim . . . 3
aliasTable . . . 5
as.data.frame_facDesign . . . 5
blocking . . . 6
cg . . . 7
cg_HistChart . . . 9
cg_RunChart . . . 11
cg_ToleranceChart . . . 13
code2real . . . 15
confounds . . . 15
contourPlot . . . 16
contourPlot3 . . . 18
desirability . . . 20
desirability.c . . . 21
desOpt . . . 23
dgamma3 . . . 24
Distr . . . 25
DistrCollection . . . 28
distribution . . . 31
dlnorm3 . . . 32
doeFactor . . . 33
dotPlot . . . 35
dweibull3 . . . 36
facDesign . . . 37
facDesign.c . . . 39
FitDistr . . . 44
fracChoose . . . 46
fracDesign . . . 46
gageLin . . . 48
gageLinDesign . . . 49
gageRR . . . 50
gageRR.c . . . 51
gageRRDesign . . . 61
interactionPlot . . . 62
mixDesign . . . 63
mixDesign.c . . . 65
MSALinearity . . . 68
mvPlot . . . 69
normalPlot . . . 71
oaChoose . . . 73
optimum . . . 74
overall . . . 75
paretoChart . . . 76
paretoPlot . . . 78
pbDesign . . . 80
pbDesign.c . . . 81

pbFactor 83
pcr 84
pgamma3 87
plnorm3 88
ppPlot 89
print_adtest 91
pweibull3 92
qgamma3 93
qlnorm3 94
qqPlot 95
qweibull3 97
randomize 98
rsmChoose 99
rsmDesign 100
simProc 101
snPlot 102
starDesign 103
steepAscent 105
steepAscent.c 106
summaryFits 107
taguchiChoose 108
taguchiDesign 109
taguchiDesign.c 111
taguchiFactor 114
wirePlot 115
wirePlot3 118

Index **120**

`adSim` *adSim: Bootstrap-based Anderson-Darling Test for Univariate*

Description

Performs the Anderson-Darling test for univariate distributions with an option for bootstrap-based p-value determination. It also allows p-value determination using tabled critical values.

Usage

```
adSim(x, distribution = "normal", b = 10000)
```

Arguments

`x`	A numeric vector.
`distribution`	A character string specifying the distribution to test. Recognized distributions include `` `cauchy` ``, `` `exponential` ``, `` `gumbel` ``, `` `gamma` ``, `` `log-normal` ``, `` `lognormal` ``, `` `logistic` ``, `` `normal` ``, and `` `weibull` ``.

b A numeric value indicating the number of bootstraps to perform. Allowed values range from 1000 to 1,000,000. If b is set to NA, the Anderson-Darling test will be applied without simulation. Note that higher values of b can significantly increase computation time, potentially taking hours depending on the distribution, sample size, and computer system.

Details

The function first estimates the parameters for the tested distribution, typically using Maximum-Likelihood Estimation (MLE) via the FitDistr function. For normal and log-normal distributions, parameters are estimated by the mean and standard deviation. Cauchy distribution parameters are fitted by the sums of the weighted order statistic when using tabled critical values. The Anderson-Darling statistic is then calculated based on these estimated parameters.

Parametric bootstrapping generates the distribution of the Anderson-Darling test statistic, which is used to determine the p-value. This simulation-based Anderson-Darling distribution and its corresponding critical values for selected quantiles can be printed. If simulation is not performed, a p-value is obtained from tabled critical values, although no exact expressions exist except for the log-normal, normal, and exponential distributions.

Value

A list containing the following components:

distribution The distribution for which the Anderson-Darling test was applied.

parameter_estimation The estimated parameters for the distribution.

Anderson_Darling The value of the Anderson-Darling test statistic.

p_value The corresponding p-value, either simulated or from tabled values.

critical_values The critical values corresponding to the 0.75, 0.90, 0.95, 0.975, and 0.99 quantiles, either simulated or from tables.

simAD The bootstrap-based Anderson-Darling distribution.

Examples

```
x <- rnorm(25, 32, 2)
adSim(x)
adSim(x, "logistic", 2000)
adSim(x, "cauchy", NA)
adSim(x, "exponential", 2000)
adSim(x, "gumbel", 2000)
```

aliasTable — *aliasTable: Display an alias table*

Description

This function generates an alias table for a factorial design object.

Usage

```
aliasTable(fdo, degree, print = TRUE)
```

Arguments

fdo	An object of class facDesign.c.
degree	Numeric value specifying the degree of interaction i.e. degree=3 means up to threeway interactions.
print	If TRUE, the alias table will be printed. By default print is set to TRUE.

Value

The function aliasTable returns a matrix indicating the aliased effects.

See Also

fracDesign, fracChoose

Examples

```
# Create a fractional factorial design
dfrac <- fracDesign(k = 3, gen = "C = AB")
# Display the alias table for the fractional factorial design
aliasTable(dfrac)
```

as.data.frame_facDesign — *as.data.frame_facDesign: Coerce to a data.frame*

Description

Converts an object of class facDesign.c into a data frame.

Usage

```
as.data.frame_facDesign(dfac)
```

Arguments

dfac	An object of class facDesign.c that you want to convert to a data frame.

Value

The function as.data.frame_facDesign returns a data frame.

Examples

```
fdo <- fracDesign(k = 2, replicates = 3, centerCube = 4)
as.data.frame_facDesign(fdo)
```

---

blocking *blocking: Blocking*

Description

Blocks a given factorial or response surface design.

Usage

```
blocking(fdo, blocks, random.seed, useTable = "rsm", gen)
```

Arguments

fdo	An object of class facDesign.c.
blocks	Numeric value giving the number of blocks.
random.seed	Numeric value to generate repeatable results for randomization within blocks.
useTable	Character indicating which table to use. The following options will be accepted: • `rms`: table from reference • `calc`: table calculated by package
gen	Giving the generator that will be used.

Value

The function blocking returns an object of class facDesign.c with blocking structure.

See Also

facDesign.

Examples

```
# Example 1
#Create a 2^3 full factorial design
fdo <- facDesign(k = 3)
# Apply blocking to the design with 2 blocks
blocking(fdo, 2)

# Example 2
#Create a response surface design for 3 factors
fdo <- rsmDesign(k = 3)
# Apply blocking to the design with 3 blocks (1 block for star part and 2 blocks for the cube part)
blocking(fdo, 3)
```

---

cg *cg: Function to calculate and visualize the gage capability.*

Description

Function visualize the given values of measurement in a run chart and in a histogram. Furthermore the `centralized` `Gage` `potential` `index` `Cg` and the `non-centralized` `Gage` `Capability` `index` `Cgk` are calculated and displayed.

Usage

```
cg(
  x,
  target,
  tolerance,
  ref.interval,
  facCg,
  facCgk,
  n = 0.2,
  col,
  pch,
  xlim,
  ylim,
  conf.level = 0.95
)
```

Arguments

`x`	A vector containing the measured values.
`target`	A numeric value giving the expected target value for the x-values.
`tolerance`	Vector of length 2 giving the lower and upper specification limits.

`ref.interval` Numeric value giving the confidence interval on which the calculation is based. By default it is based on 6 sigma methodology. Regarding the normal distribution this relates to `pnorm(3) - pnorm(-3)` which is exactly 99.73002 percent. If the calculation is based on another sigma value `ref.interval` needs to be adjusted. To give an example: If the sigma-level is given by 5.15 the `ref.interval` relates to `pnorm(5.15/2)-pnorm(-5.15/2)` which is exactly 0.989976 percent.

`facCg` Numeric value as a factor for the calculation of the gage potential index. The default value for `facCg` is `0.2`.

`facCgk` Numeric value as a factor for the calculation of the gage capability index. The default value for `facCgk` is `0.1`.

`n` Numeric value between `0` and `1` giving the percentage of the tolerance field (values between the upper and lower specification limits given by `tolerance`) where the values of `x` should be positioned. Limit lines will be drawn. Default value is `0.2`.

`col` Character or numeric value specifying the color of the curve in the run chart. Default is `` `black` ``.

`pch` Numeric or character specifying the plotting symbol. Default is `19` (filled circle).

`xlim` Numeric vector of length 2 specifying the limits for the x-axis. Default is `NULL` which means the limits are set automatically.

`ylim` Numeric vector of length 2 specifying the limits for the y-axis. Default is `NULL` which means the limits are set automatically.

`conf.level` Confidence level for internal `t.test` checking the significance of the bias between `target` and mean of `x`. The default value is `0.95`. The result of the `t.test` is shown in the histogram on the left side.

Details

The calculation of the potential and actual gage capability are based on the following formula:

- `Cg = (facCg * tolerance[2]-tolerance[1])/ref.interval`
- `Cgk = (facCgk * abs(target-mean(x))/(ref.interval/2)`

If the usage of the historical process variation is preferred the values for the tolerance `tolerance` must be adjusted manually. That means in case of the 6 sigma methodology for example, that `tolerance = 6 * sigma[process]`.

Value

The function `cg` returns a list of numeric values. The first element contains the calculated centralized gage potential index `Cg` and the second contains the non-centralized gage capability index `Cgk`.

See Also

`cg_RunChart`, `cg_HistChart`, `cg_ToleranceChart`.

Examples

```
x <- c(9.991, 10.013, 10.001, 10.007, 10.010, 10.013, 10.008,9.992,
       10.017, 10.005, 10.005, 10.002, 10.017, 10.005, 10.002, 9.996,
       10.011, 10.009, 10.006, 10.008, 10.003, 10.002, 10.006, 10.010, 10.013)

cg(x = x, target = 10.003, tolerance = c(9.903, 10.103))
```

---

cg_HistChart *cg_HistChart*

Description

Function visualize the given values of measurement in a histogram

Usage

```
cg_HistChart(
  x,
  target,
  tolerance,
  ref.interval,
  facCg,
  facCgk,
  n = 0.2,
  col,
  xlim,
  ylim,
  main,
  conf.level = 0.95,
  cgOut = TRUE
)
```

Arguments

`x`	A vector containing the measured values.
`target`	A numeric value giving the expected target value for the x-values.
`tolerance`	Vector of length 2 giving the lower and upper specification limits.
`ref.interval`	Numeric value giving the confidence interval on which the calculation is based. By default it is based on 6 sigma methodology. Regarding the normal distribution this relates to `pnorm(3) - pnorm(-3)` which is exactly 99.73002 percent. If the calculation is based on another sigma value `ref.interval` needs to be adjusted. To give an example: If the sigma-level is given by 5.15 the `ref.interval` relates to `pnorm(5.15/2)-pnorm(-5.15/2)` which is exactly 0.989976 percent.
`facCg`	Numeric value as a factor for the calculation of the gage potential index. The default Value for `facCg` is `0.2`.

facCgk	Numeric value as a factor for the calculation of the gage capability index. The default value for facCgk is 0.1.
n	Numeric value between 0 and 1 giving the percentage of the tolerance field (values between the upper and lower specification limits given by tolerance) where the values of x should be positioned. Limit lines will be drawn. Default value is 0.2.
col	Character or numeric value specifying the color of the histogram. Default is `black`.
xlim	Numeric vector of length 2 specifying the limits for the x-axis. Default is NULL which means the limits are set automatically.
ylim	Numeric vector of length 2 specifying the limits for the y-axis. Default is NULL which means the limits are set automatically.
main	Character string specifying the title of the plot. Default is `Histogram of x - target`.
conf.level	Confidence level for internal t.test checking the significance of the bias between target and mean of x. The default value is 0.95.
cgOut	Logical value deciding whether the Cg and Cgk values should be plotted in a legend. Default is TRUE.

Details

The calculation of the potential and actual gage capability are based on the following formulae:

- Cg = (facCg * tolerance[2]-tolerance[1])/ref.interval
- Cgk = (facCgk * abs(target-mean(x))/(ref.interval/2)

If the usage of the historical process variation is preferred the values for the tolerance tolerance must be adjusted manually. That means in case of the 6 sigma methodology for example, that tolerance = 6 * sigma[process].

Value

The function cg_HistChart returns a list of numeric values. The first element contains the calculated centralized gage potential index Cg and the second contains the non-centralized gage capability index Cgk.

See Also

cg_RunChart, cg_ToleranceChart, cg

Examples

```
x <- c(9.991, 10.013, 10.001, 10.007, 10.010, 10.013, 10.008,9.992,
       10.017, 10.005, 10.005, 10.002, 10.017, 10.005, 10.002, 9.996,
       10.011, 10.009, 10.006, 10.008, 10.003, 10.002, 10.006, 10.010, 10.013)

cg_HistChart(x = x, target = 10.003, tolerance = c(9.903, 10.103))
```

cg_RunChart *cg_RunChart*

Description

Function visualize the given values of measurement in a Run Chart

Usage

```
cg_RunChart(
  x,
  target,
  tolerance,
  ref.interval,
  facCg,
  facCgk,
  n = 0.2,
  col = "black",
  pch = 19,
  xlim = NULL,
  ylim = NULL,
  main = "Run Chart",
  conf.level = 0.95,
  cgOut = TRUE
)
```

Arguments

`x`	A vector containing the measured values.
`target`	A numeric value giving the expected target value for the x-values.
`tolerance`	Vector of length 2 giving the lower and upper specification limits.
`ref.interval`	Numeric value giving the confidence interval on which the calculation is based. By default it is based on 6 sigma methodology. Regarding the normal distribution this relates to `pnorm(3) - pnorm(-3)` which is exactly 99.73002 percent. If the calculation is based on another sigma value `ref.interval` needs to be adjusted. To give an example: If the sigma-level is given by 5.15 the `ref.interval` relates to `pnorm(5.15/2)-pnorm(-5.15/2)` which is exactly 0.989976 percent.
`facCg`	Numeric value as a factor for the calculation of the gage potential index. The default Value for `facCg` is `0.2`.
`facCgk`	Numeric value as a factor for the calculation of the gage capability index. The default value for `facCgk` is `0.1`.
`n`	Numeric value between `0` and `1` giving the percentage of the tolerance field (values between the upper and lower specification limits given by `tolerance`) where the values of `x` should be positioned. Limit lines will be drawn. Default value is `0.2`.

col	Character or numeric value specifying the color of the curve in the run chart. Default is `black`.
pch	Numeric or character specifying the plotting symbol. Default is 19 (filled circle).
xlim	Numeric vector of length 2 specifying the limits for the x-axis. Default is NULL which means the limits are set automatically.
ylim	Numeric vector of length 2 specifying the limits for the y-axis. Default is NULL which means the limits are set automatically.
main	Character string specifying the title of the plot. Default is `Run Chart`.
conf.level	Confidence level for internal t.test checking the significance of the bias between target and mean of x. The default value is 0.95. The result of the t.test is shown in the histogram on the left side.
cgOut	Logical value deciding whether the Cg and Cgk values should be plotted in a legend. Default is TRUE.

Details

The calculation of the potential and actual gage capability are based on the following formulae:

- Cg = (facCg * tolerance[2]-tolerance[1])/ref.interval
- Cgk = (facCgk * abs(target-mean(x))/(ref.interval/2)

If the usage of the historical process variation is preferred the values for the tolerance tolerance must be adjusted manually. That means in case of the 6 sigma methodology for example, that tolerance = 6 * sigma[process].

Value

The function cg_RunChart returns a list of numeric values. The first element contains the calculated centralized gage potential index Cg and the second contains the non-centralized gage capability index Cgk.

See Also

cg_HistChart, cg_ToleranceChart, cg

Examples

```
x <- c(9.991, 10.013, 10.001, 10.007, 10.010, 10.013, 10.008,9.992,
       10.017, 10.005, 10.005, 10.002, 10.017, 10.005, 10.002, 9.996,
       10.011, 10.009, 10.006, 10.008, 10.003, 10.002, 10.006, 10.010, 10.013)

cg_RunChart(x = x, target = 10.003, tolerance = c(9.903, 10.103))
```

cg_ToleranceChart *cg_ToleranceChart*

Description

Function visualize the given values of measurement in a Tolerance View.

Usage

```
cg_ToleranceChart(
  x,
  target,
  tolerance,
  ref.interval,
  facCg,
  facCgk,
  n = 0.2,
  col,
  pch,
  xlim,
  ylim,
  main,
  conf.level = 0.95,
  cgOut = TRUE
)
```

Arguments

`x`	A vector containing the measured values.
`target`	A numeric value giving the expected target value for the x-values.
`tolerance`	Vector of length 2 giving the lower and upper specification limits.
`ref.interval`	Numeric value giving the confidence interval on which the calculation is based. By default it is based on 6 sigma methodology. Regarding the normal distribution this relates to `pnorm(3) - pnorm(-3)` which is exactly 99.73002 percent. If the calculation is based on another sigma value `ref.interval` needs to be adjusted. To give an example: If the sigma-level is given by 5.15 the `ref.interval` relates to `pnorm(5.15/2)-pnorm(-5.15/2)` which is exactly 0.989976 percent.
`facCg`	Numeric value as a factor for the calculation of the gage potential index. The default value for `facCg` is `0.2`.
`facCgk`	Numeric value as a factor for the calculation of the gage capability index. The default value for `facCgk` is `0.1`.
`n`	Numeric value between `0` and `1` giving the percentage of the tolerance field (values between the upper and lower specification limits given by `tolerance`) where the values of `x` should be positioned. Limit lines will be drawn. Default value is `0.2`.

col	Character or numeric value specifying the color of the line and points in the tolerance view. Default is `black`.
pch	Numeric or character specifying the plotting symbol. Default is 19 (filled circle).
xlim	Numeric vector of length 2 specifying the limits for the x-axis. Default is NULL which means the limits are set automatically.
ylim	Numeric vector of length 2 specifying the limits for the y-axis. Default is NULL which means the limits are set automatically.
main	Character string specifying the title of the plot. Default is `Tolerance View`.
conf.level	Confidence level for internal t.test checking the significance of the bias between target and mean of x. The default value is 0.95.
cgOut	Logical value deciding whether the Cg and Cgk values should be plotted in a legend. Default is TRUE.

Details

The calculation of the potential and actual gage capability are based on the following formulae:

- Cg = (facCg * tolerance[2]-tolerance[1])/ref.interval
- Cgk = (facCgk * abs(target-mean(x))/(ref.interval/2)

If the usage of the historical process variation is preferred the values for the tolerance tolerance must be adjusted manually. That means in case of the 6 sigma methodology for example, that tolerance = 6 * sigma[process].

Value

The function cg_ToleranceChart returns a list of numeric values. The first element contains the calculated centralized gage potential index Cg and the second contains the non-centralized gage capability index Cgk.

See Also

cg_RunChart, cg_HistChart, cg

Examples

```
x <- c(9.991, 10.013, 10.001, 10.007, 10.010, 10.013, 10.008,9.992,
       10.017, 10.005, 10.005, 10.002, 10.017, 10.005, 10.002, 9.996,
       10.011, 10.009, 10.006, 10.008, 10.003, 10.002, 10.006, 10.010, 10.013)

cg_ToleranceChart(x = x, target = 10.003, tolerance = c(9.903, 10.103))
```

code2real — *code2real: Coding*

Description

Function to calculate the real value of a coded value.

Usage

```
code2real(low, high, codedValue)
```

Arguments

low	Numeric value giving the lower boundary.
high	Numeric value giving the higher boundary.
codedValue	Numeric value giving the coded value that will be calculated.

Value

The function return a real value of a coded value

Examples

```
code2real(160, 200, 0)
```

confounds — *confounds: Confounded Effects*

Description

Function to display confounded effects of a fractional factorial design in a human readable way.

Usage

```
confounds(x, depth = 2)
```

Arguments

x	An object of class facDesign.c.
depth	numeric value - up to depth-way confounded interactions are printed

Value

The function returns a summary of the factors confounded.

Examples

```
vp.frac = fracDesign(k = 4, gen = "D=ABC")
confounds(vp.frac,depth=5)
```

---

contourPlot *contourPlot: Contour Plot*

Description

Creates a contour diagram for an object of class facDesign.c.

Usage

```
contourPlot(
  x,
  y,
  z,
  data = NULL,
  xlim,
  ylim,
  main,
  xlab,
  ylab,
  form = "fit",
  col = 1,
  steps,
  fun,
  plot = TRUE,
  show.scale = TRUE
)
```

Arguments

x	Name providing the Factor A for the plot.
y	Name providing the Factor B for the plot.
z	Name giving the Response variable.
data	Needs to be an object of class facDesign.c and contains the names of x, y, z.
xlim	Vector giving the range of the x-axis.
ylim	Vector giving the range of the y-axis.
main	Character string: title of the plot.
xlab	Character string: label for the x-axis.
ylab	Character string: label for the y-axis.
form	A character string or a formula with the syntax ‘y~ x+y + x*y‘. If form is a character it has to be one out of the following:

- `quadratic`
- `full`
- `interaction`
- `linear`
- `fit`

`fit` takes the formula from the fit in the facDesign.c object fdo. Quadratic or higher orders should be given as I(Variable^2). By default form is set as `fit`.

col — A predefined (1, 2, 3 or 4) or self defined colorRampPalette or color to be used (i.e. `red`).

steps — Number of grid points per factor. By default steps = 25.

fun — Function to be applied to z desirability.

plot — Logical value indicating whether to display the plot. Default is TRUE.

show.scale — Logical value indicating whether to display the color scale on the plot. Default is TRUE.

Value

The function contourPlot returns an invisible list containing:

- x - locations of grid lines for x at which the values in z are measured.
- y - locations of grid lines for y at which the values in z are measured.
- z - a matrix containing the values of z to be plotted.
- plot - The generated plot.

See Also

wirePlot, paretoChart

Examples

```
fdo = rsmDesign(k = 3, blocks = 2)
fdo$.response(data.frame(y = rnorm(fdo$nrow())))

#I - display linear fit
contourPlot(A,B,y, data = fdo, form = "linear")
#II - display full fit (i.e. effect, interactions and quadratic effects
contourPlot(A,B,y, data = fdo, form = "full")
#III - display a fit specified before
fdo$set.fits(fdo$lm(y ~ B + I(A^2)))
contourPlot(A,B,y, data = fdo, form = "fit")
#IV - display a fit given directly
contourPlot(A,B,y, data = fdo, form = "y ~ A*B + I(A^2)")
#V - display a fit using a different colorRamp
contourPlot(A,B,y, data = fdo, form = "full", col = 2)
#VI - display a fit using a self defined colorRamp
myColour = colorRampPalette(c("green", "gray","blue"))
contourPlot(A,B,y, data = fdo, form = "full", col = myColour)
```

contourPlot3 *contourPlot3: Ternary plot*

Description

This function creates a ternary plot (contour plot) for mixture designs (i.e. object of class `mixDesign`).

Usage

```
contourPlot3(
  x,
  y,
  z,
  response,
  data = NULL,
  main,
  xlab,
  ylab,
  zlab,
  form = "linear",
  col = 1,
  col.text,
  axes = TRUE,
  steps,
  plot = TRUE,
  show.scale = TRUE
)
```

Arguments

`x`	Factor 1 of the `mixDesign` object.
`y`	Factor 2 of the `mixDesign` object.
`z`	Factor 3 of the `mixDesign` object.
`response`	the response of the `mixDesign` object.
`data`	The `mixDesign` object from which x,y,z and the response are taken.
`main`	Character string specifying the main title of the plot.
`xlab`	Character string specifying the label for the x-axis.
`ylab`	Character string specifying the label for the y-axis.
`zlab`	Character string specifying the label for the z-axis.
`form`	A character string or a formula with the syntax ‘y ~ A + B + C‘. If form is a character string, it has to be one of the following: • ‘linear‘ • ‘quadratic‘

How the form influences the output is described in the reference listed below. By default, `form` is set to 'linear'.

`col` A predefined value (1, 2, 3, or 4) or a self-defined `colorRampPalette` specifying the colors to be used in the plot.

`col.text` A character string specifying the color of the axis labels. The default value `col.text` is '1'.

`axes` A logical value specifying whether the axes should be plotted. By default, `axes` is set to `TRUE`.

`steps` A numeric value specifying the resolution of the plot, i.e., the number of rows for the square matrix, which also represents the number of grid points per factor. By default, `steps` is set to 25.

`plot` Logical value indicating whether to display the plot. Default is `TRUE`.

`show.scale` Logical value indicating whether to display the color scale on the plot. Default is `TRUE`.

Value

The function `contourPlot3` returns an invisible list containing:

- mat - A matrix containing the response values as NA's and numerics.
- plot - The generated plot.

See Also

`mixDesign.c`, `mixDesign`, `wirePlot3`.

Examples

```
mdo = mixDesign(3,2, center = FALSE, axial = FALSE, randomize = FALSE,
                replicates  = c(1,1,2,3))
mdo$names(c("polyethylene", "polystyrene", "polypropylene"))
mdo$units("percent")
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)
mdo$.response(elongation)
contourPlot3(A, B, C, elongation, data = mdo, form = "linear")
contourPlot3(A, B, C, elongation, data = mdo, form = "quadratic", col = 2)
contourPlot3(A, B, C, elongation, data = mdo,
             form = "elongation ~ I(A^2) - B:A + I(C^2)",
             col = 3, axes = FALSE)
contourPlot3(A, B, C, elongation, data = mdo,
             form = "quadratic",
             col = c("yellow", "white", "red"),
             axes = FALSE)
```

desirability *desirability: Desirability Function.*

Description

Creates desirability functions for use in the optimization of multiple responses.

Usage

```
desirability(
  response,
  low,
  high,
  target = "max",
  scale = c(1, 1),
  importance = 1
)
```

Arguments

response	Name of the response.
low	Lowest acceptable value for the response.
high	Highest acceptable value for the response.
target	Desired target value of the response. `target` can be `` `max` ``, `` `min` ``, or any specific numeric value.
scale	Numeric value giving the scaling factors for one and two-sided transformations. Default is `c(1, 1)`.
importance	A value ranging from 0.1 to 10, used to calculate a weighted importance, i.e., with importances 1, 2, and 4, D = [(d1)^1, (d2)^2, (d3)^4]^(1/7). Default is '1'.

Details

For a product to be developed, different values of responses are desired, leading to multiple response optimization. Minimization, maximization, as well as a specific target value, are defined using desirability functions. A desirability function transforms the values of a response into [0,1], where 0 stands for a non-acceptable value of the response and 1 for values where higher/lower (depending on the direction of the optimization) values of the response have little merit. This function builds upon the desirability functions specified by Harrington (1965) and the modifications by Derringer and Suich (1980) and Derringer (1994). Castillo, Montgomery, and McCarville (1996) further extended these functions, but these extensions are not implemented in this version.

Value

This function returns a `desirability.c` object.

See Also

overall, optimum

Examples

```
# Example 1: Maximization of a response
# Define a desirability for response y where higher values of y are better
# as long as the response is smaller than high
d = desirability(y, low = 6, high = 18, target = "max")
# Show and plot the desirability function
d
plot(d)

# Example 2: Minimization of a response including a scaling factor
# Define a desirability for response y where lower values of y are better
# as long as the response is higher than low
d = desirability(y, low = 6, high = 18, scale = c(2), target = "min")
# Show and plot the desirability function
d
plot(d)

# Example 3: Specific target of a response is best including a scaling factor
# Define a desirability for response y where desired value is at 8 and
# values lower than 6 as well as values higher than 18 are not acceptable
d = desirability(y, low = 6, high = 18, scale = c(0.5, 2), target = 12)
# Show and plot the desirability function
d
plot(d)

# Example 4:
y1 <- c(102, 120, 117, 198, 103, 132, 132, 139, 102, 154, 96, 163, 116, 153,
        133, 133, 140, 142, 145, 142)
y2 <- c(470, 410, 570, 240, 640, 270, 410, 380, 590, 260, 520, 380, 520, 290,
        380, 380, 430, 430, 390, 390)
d1 <- desirability(y1, 120, 170, scale = c(1, 1), target = "max")
d3 <- desirability(y2, 400, 600, target = 500)
d1
plot(d1)
d3
plot(d3)
```

---

desirability.c *desirability-class: Class ‘desirability‘*

Description

A class representing the desirability metrics for responses in a design.

Public fields

`response` A numeric vector specifying the responses for which desirability is calculated.

`low` A numeric vector representing the lower bounds of the desirable range for each response.

`high` A numeric vector representing the upper bounds of the desirable range for each response.

`target` A numeric vector or character string indicating the target values or goals for each response.

`scale` A numeric vector specifying the scaling factors used in the desirability calculation.

`importance` A numeric vector indicating the importance of each response in the desirability calculation.

Methods

Public methods:

- `desirability.c$new()`
- `desirability.c$print()`
- `desirability.c$plot()`
- `desirability.c$clone()`

Method `new()`**:** Initializes a new `desirability.c` object with specified parameters.

Usage:

```
desirability.c$new(
  response = NULL,
  low = NULL,
  high = NULL,
  target = NULL,
  scale = NULL,
  importance = NULL
)
```

Arguments:

`response` A numeric or character vector specifying the responses for which desirability is calculated.

`low` A numeric vector representing the lower bounds of the desirable range for each response.

`high` A numeric vector representing the upper bounds of the desirable range for each response.

`target` A numeric vector or character string indicating the target values or goals for each response.

`scale` A numeric vector specifying the scaling factors used in the desirability calculation.

`importance` A numeric vector indicating the importance of each response in the desirability calculation.

Method `print()`**:** Prints the details of a `desirability.c` object.

Usage:

```
desirability.c$print()
```

Method `plot()`**:** Plots the desirability functions based on the specified parameters.

Usage:

```
desirability.c$plot(scale, main, xlab, ylab, line.width, col, numPoints = 500)
```

Arguments:

`scale` A numeric vector specifying the scaling factors used in the plot.

`main` A character string specifying the main title of the plot.

`xlab` A character string specifying the label for the x-axis.

`ylab` A character string specifying the label for the y-axis.

`line.width` A numeric value specifying the width of the plot lines.

`col` A vector of colors for the plot lines.

`numPoints` An integer specifying the number of points to plot (default is 500).

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
desirability.c$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`desirability`, `overall`, `optimum`

`desOpt` *desOpt-class: Class 'desOpt'*

Description

The `desOpt` class represents an object that stores optimization results for factorial design experiments. It includes coded and real factors, responses, desirabilities, overall desirability, and the design object.

Public fields

`facCoded` A list containing the coded values for the factors in the design.

`facReal` A list containing the real (actual) values for the factors in the design.

`responses` A list of response variables obtained from the design.

`desirabilities` A list of desirability scores for each response variable.

`overall` Numeric value representing the overall desirability score.

`all` A data frame containing all the relevant data from the design and optimization process.

`fdo` The factorial design object used in the optimization process.

Methods

Public methods:

- desOpt$as.data.frame()
- desOpt$print()
- desOpt$clone()

Method `as.data.frame()`**:** Convert the object to a data frame.

Usage:

```
desOpt$as.data.frame()
```

Method `print()`**:** Print a summary of the object.

Usage:

```
desOpt$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
desOpt$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

optimum, facDesign, desirability

---

dgamma3 *dgamma3: The gamma Distribution (3 Parameter)*

Description

Density function, distribution function, and quantile function for the Gamma distribution.

Usage

```
dgamma3(x, shape, scale, threshold)
```

Arguments

x	A numeric vector of quantiles.
shape	The shape parameter, default is 1.
scale	The scale parameter, default is 1.
threshold	The threshold parameter, default is 0.

Details

The Gamma distribution with `scale` parameter alpha, `shape` parameter c, and `threshold` parameter zeta has a density given by:

$$f(x) = \frac{c}{\alpha}\left(\frac{x-\zeta}{\alpha}\right)^{c-1}\exp\left(-\left(\frac{x-\zeta}{\alpha}\right)^{c}\right)$$

The cumulative distribution function is given by:

$$F(x) = 1 - \exp\left(-\left(\frac{x-\zeta}{\alpha}\right)^{c}\right)$$

Value

`dgamma3` gives the density, `pgamma3` gives the distribution function, and `qgamma3` gives the quantile function.

Examples

```
dgamma3(x = 1, scale = 1, shape = 5, threshold = 0)
temp <- pgamma3(q = 1, scale = 1, shape = 5, threshold = 0)
temp
qgamma3(p = temp, scale = 1, shape = 5, threshold = 0)
```

`Distr` *Distr-class: Class 'Distr'*

Description

R6 Class for Distribution Objects

Public fields

`x` Numeric vector of data values.

`name` Character string representing the name of the distribution.

`parameters` List of parameters for the distribution.

`sd` Numeric value representing the standard deviation of the distribution.

`n` Numeric value representing the sample size.

`loglik` Numeric value representing the log-likelihood.

Methods

Public methods:

- Distr$new()
- Distr$plot()
- Distr$clone()

Method `new()`**:** Initialize the fiels of the ‘Distribution‘ object

Usage:

```
Distr$new(x, name, parameters, sd, n, loglik)
```

Arguments:

`x` Numeric vector of data values.

`name` Character string representing the name of the distribution.

`parameters` List of parameters for the distribution.

`sd` Numeric value representing the standard deviation of the distribution.

`n` Numeric value representing the sample size.

`loglik` Numeric value representing the log-likelihood.

Method `plot()`**:** Plot the distribution with histogram and fitted density curve.

Usage:

```
Distr$plot(
  main = NULL,
  xlab = NULL,
  xlim = NULL,
  xlim.t = TRUE,
  ylab = NULL,
  line.col = "red",
  fill.col = "lightblue",
  border.col = "black",
  box = TRUE,
  line.width = 1
)
```

Arguments:

`main` Character string for the main title of the plot. Defaults to the name of the distribution.

`xlab` Character string for the x-axis label. Defaults to "x".

`xlim` Numeric vector specifying the x-axis limits.

`xlim.t` Logical value specifyind to change the xlim default.

`ylab` Character string for the y-axis label. Defaults to "Density".

`line.col` Character string for the color of the plot line. Default is "red".

`fill.col` Character string for the color of the fill histogram plot line. Default is "lightblue".

`border.col` Character string for the color of the border of the fill histogram plot line. Default is "black".

`box` Logical value indicating whether to draw a box with the parameters in the plot. Default is TRUE.

`line.width` Numeric value specifying the width of the plot line. Default is 1.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Distr$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

distribution, FitDistr, DistrCollection

Examples

```
# Normal
set.seed(123)
data1 <- rnorm(100, mean = 5, sd = 2)
parameters1 <- list(mean = 5, sd = 2)
distr1 <- Distr$new(x = data1, name = "normal", parameters = parameters1,
                    sd = 2, n = 100, loglik = -120)
distr1$plot()

# Log-normal
data2 <- rlnorm(100, meanlog = 1, sdlog = 0.5)
parameters2 <- list(meanlog = 1, sdlog = 0.5)
distr2 <- Distr$new(x = data2, name = "log-normal", parameters = parameters2,
                    sd = 0.5, n = 100, loglik = -150)
distr2$plot()

# Geometric
data3 <- rgeom(100, prob = 0.3)
parameters3 <- list(prob = 0.3)
distr3 <- Distr$new(x = data3, name = "geometric", parameters = parameters3,
                    sd = sqrt((1 - 0.3) / (0.3^2)), n = 100, loglik = -80)
distr3$plot()

# Exponential
data4 <- rexp(100, rate = 0.2)
parameters4 <- list(rate = 0.2)
distr4 <- Distr$new(x = data4, name = "exponential", parameters = parameters4,
                    sd = 1 / 0.2, n = 100, loglik = -110)
distr4$plot()

# Poisson
data5 <- rpois(100, lambda = 3)
parameters5 <- list(lambda = 3)
distr5 <- Distr$new(x = data5, name = "poisson", parameters = parameters5,
                    sd = sqrt(3), n = 100, loglik = -150)
distr5$plot()

# Chi-square
```

```
data6 <- rchisq(100, df = 5)
parameters6 <- list(df = 5)
distr6 <- Distr$new(x = data6, name = "chi-squared", parameters = parameters6,
                    sd = sqrt(2 * 5), n = 100, loglik = -130)
distr6$plot()

# Logistic
data7 <- rlogis(100, location = 0, scale = 1)
parameters7 <- list(location = 0, scale = 1)
distr7 <- Distr$new(x = data7, name = "logistic", parameters = parameters7,
                    sd = 1 * sqrt(pi^2 / 3), n = 100, loglik = -140)
distr7$plot()

# Gamma
data8 <- rgamma(100, shape = 2, rate = 0.5)
parameters8 <- list(shape = 2, rate = 0.5)
distr8 <- Distr$new(x = data8, name = "gamma", parameters = parameters8,
                    sd = sqrt(2 / (0.5^2)), n = 100, loglik = -120)
distr8$plot()

# f
data9 <- rf(100, df1 = 5, df2 = 10)
parameters9 <- list(df1 = 5, df2 = 10)
df1 <- 5
df2 <- 10
distr9 <- Distr$new(x = data9, name = "f", parameters = parameters9,
                    sd = sqrt(((df2^2 * (df1 + df2 - 2)) /
                               (df1 * (df2 - 2)^2 * (df2 - 4)))),
                    n = 100, loglik = -150)
distr9$plot()

# t
data10 <- rt(100, df = 10)
parameters10 <- list(df = 10)
distr10 <- Distr$new(x = data10, name = "t", parameters = parameters10,
                     sd = sqrt(10 / (10 - 2)), n = 100, loglik = -120)
distr10$plot()

# negative binomial
data11 <- rnbinom(100, size = 5, prob = 0.3)
parameters11 <- list(size = 5, prob = 0.3)
distr11 <- Distr$new(x = data11, name = "negative binomial",
                     parameters = parameters11,
                     sd = sqrt(5 * (1 - 0.3) / (0.3^2)),
                     n = 100, loglik = -130)
distr11$plot()
```

`DistrCollection` *DistrCollection-class: Class 'DistrCollection'*

Description

R6 Class for Managing a Collection of Distribution Objects

Public fields

`distr` List of `Distr` objects.

Methods

Public methods:

- `DistrCollection$new()`
- `DistrCollection$add()`
- `DistrCollection$get()`
- `DistrCollection$print()`
- `DistrCollection$summary()`
- `DistrCollection$plot()`
- `DistrCollection$clone()`

Method `new()`**:** Initialize the fields of the `DistrCollection` object.

Usage:

```
DistrCollection$new()
```

Method `add()`**:** Add a `Distr` object to the collection.

Usage:

```
DistrCollection$add(distr)
```

Arguments:

`distr` A `Distr` object to add to the collection.

Method `get()`**:** Get a `Distr` object from the collection by its index.

Usage:

```
DistrCollection$get(i)
```

Arguments:

`i` Integer index of the `Distr` object to retrieve.

Returns: A `Distr` object.

Method `print()`**:** Print the summary of all distributions in the collection.

Usage:

```
DistrCollection$print()
```

Method `summary()`**:** Summarize the goodness of fit for all distributions in the collection.

Usage:

```
DistrCollection$summary()
```

Returns: A data frame with distribution names, Anderson-Darling test statistics, and p-values.

Method `plot()`**:** Plot all distributions in the collection.

Usage:

```
DistrCollection$plot(
  xlab = NULL,
  ylab = NULL,
  xlim = NULL,
  ylim = NULL,
  line.col = "red",
  fill.col = "lightblue",
  border.col = "black",
  line.width = 1,
  box = TRUE
)
```

Arguments:

`xlab` Character string for the x-axis label.

`ylab` Character string for the y-axis label.

`xlim` Numeric vector specifying the x-axis limits.

`ylim` Numeric vector specifying the y-axis limits.

`line.col` Character string for the color of the plot line. Default is "red".

`fill.col` Character string for the color of the histogram fill. Default is "lightblue".

`border.col` Character string for the color of the histogram border. Default is "black".

`line.width` Numeric value specifying the width of the plot line. Default is 1.

`box` Logical value indicating whether to draw a box with the parameters in the plot. Default is TRUE.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
DistrCollection$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`Distr`, `distribution`, `FitDistr`

Examples

```
set.seed(123)
data1 <- rnorm(100, mean = 5, sd = 2)
parameters1 <- list(mean = 5, sd = 2)
distr1 <- Distr$new(x = data1, name = "normal",
                    parameters = parameters1, sd = 2,
                    n = 100, loglik = -120)

data2 <- rpois(100, lambda = 3)
parameters2 <- list(lambda = 3)
distr2 <- Distr$new(x = data2, name = "poisson",
```

```
                    parameters = parameters2, sd = sqrt(3),
                    n = 100, loglik = -150)
collection <- DistrCollection$new()
collection$add(distr1)
collection$add(distr2)
collection$summary()
collection$plot()
```

`distribution` *distribution: Distribution*

Description

Calculates the most likely parameters for a given distribution.

Usage

```
distribution(x = NULL, distrib = "weibull", ...)
```

Arguments

`x`	Vector of distributed values from which the parameter should be determined.
`distrib`	Character string specifying the distribution of x. The function `distribution` will accept the following character strings for `distribution`: • `` `normal` `` • `` `chi-squared` `` • `` `exponential` `` • `` `logistic` `` • `` `gamma` `` • `` `weibull` `` • `` `cauchy` `` • `` `beta` `` • `` `f` `` • `` `t` `` • `` `geometric` `` • `` `poisson` `` • `` `negative binomial` `` • `` `log-normal` `` By default, `distribution` is set to `` `weibull` ``.
`...`	Additional arguments to be passed to the fitting function.

Value

`distribution()` returns an object of class `DistrCollection`.

See Also

Distr, DistrCollection

Examples

```
data1 <- rnorm(100, mean = 5, sd = 2)
distribution(data1, distrib = "normal")
```

---

dlnorm3 *dlnorm3: The Lognormal Distribution (3 Parameter)*

Description

Density function, distribution function, and quantile function for the Lognormal distribution.

Usage

```
dlnorm3(x, meanlog, sdlog, threshold)
```

Arguments

x	A numeric vector of quantiles.
meanlog, sdlog	The mean and standard deviation of the distribution on the log scale with default values of 0 and 1 respectively.
threshold	The threshold parameter, default is 0.

Details

The Lognormal distribution with meanlog parameter zeta, sdlog parameter sigma, and threshold parameter theta has a density given by:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma(x-\theta)} \exp\left(-\frac{(\log(x-\theta)-\zeta)^2}{2\sigma^2}\right)$$

The cumulative distribution function is given by:

$$F(x) = \Phi\left(\frac{\log(x-\theta)-\zeta}{\sigma}\right)$$

where Φ is the cumulative distribution function of the standard normal distribution.

Value

dlnorm3 gives the density, plnorm3 gives the distribution function, and qlnorm3 gives the quantile function.

Examples

```
dlnorm3(x = 2, meanlog = 0, sdlog = 1/8, threshold = 1)
temp <- plnorm3(q = 2, meanlog = 0, sdlog = 1/8, threshold = 1)
temp
qlnorm3(p = temp, meanlog = 0, sdlog = 1/8, threshold = 1)
```

`doeFactor` *doeFactor-class: Class ‘doeFactor‘*

Description

An R6 class representing a factor in a design of experiments (DOE).

Public fields

`low` Numeric value specifying the lower bound of the factor. Default is ‘-1‘.

`high` Numeric value specifying the upper bound of the factor. Default is ‘1‘.

`name` Character string specifying the name of the factor. Default is an empty string ``.

`unit` Character string specifying the unit of measurement for the factor. Default is an empty string ``.

`type` Character string specifying the type of the factor. Can be either `numeric` or `categorical`. Default is `numeric`.

Methods

Public methods:

- `doeFactor$attributes()`
- `doeFactor$.low()`
- `doeFactor$.high()`
- `doeFactor$.type()`
- `doeFactor$.unit()`
- `doeFactor$names()`
- `doeFactor$print()`
- `doeFactor$clone()`

Method `attributes()`**:** Get the attributes of the factor.

Usage:

```
doeFactor$attributes()
```

Method `.low()`**:** Get and set the lower bound for the factor.

Usage:

```
doeFactor$.low(value)
```

Arguments:

`value` Numeric value to set as the lower bound. If missing, the current lower bound is returned.

Method `.high()`**:** Get and set the upper bound for the factor.

Usage:

```
doeFactor$.high(value)
```

Arguments:

`value` Numeric value to set as the upper bound. If missing, the current upper bound is returned.

Method `.type()`**:** Get and set the type of the factor.

Usage:

```
doeFactor$.type(value)
```

Arguments:

`value` Character string specifying the type of the factor. Can be ‘"numeric"‘ or ‘"categorical"‘. If missing, the current type is returned.

Method `.unit()`**:** Get and set the unit of measurement for the factor.

Usage:

```
doeFactor$.unit(value)
```

Arguments:

`value` Character string specifying the unit of measurement. If missing, the current unit is returned.

Method `names()`**:** Get and set the name of the factor.

Usage:

```
doeFactor$names(value)
```

Arguments:

`value` Character string specifying the name of the factor. If missing, the current name is returned.

Method `print()`**:** Print the characteristics of the factors.

Usage:

```
doeFactor$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
doeFactor$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`taguchiFactor`

dotPlot *dotPlot: Function to create a dot plot*

Description

Creates a dot plot. For data in groups, the dot plot can be displayed stacked or in separate regions.

Usage

```
dotPlot(
  x,
  group,
  xlim,
  ylim,
  col,
  xlab,
  ylab,
  pch,
  cex,
  breaks,
  stacked = TRUE,
  main,
  showPlot = TRUE
)
```

Arguments

x	A numeric vector containing the values to be plotted.
group	(Optional) A vector for grouping the values. This determines the grouping of the data points in the dot plot.
xlim	A numeric vector of length 2 specifying the limits of the x-axis (lower and upper limits).
ylim	A numeric vector of length 2 specifying the limits of the y-axis (lower and upper limits).
col	A vector containing numeric values or strings specifying the colors for the different groups in the dot plot.
xlab	A title for the x-axis.
ylab	A title for the y-axis.
pch	A vector of integers specifying the symbols or a single character to be used for plotting points for the different groups in the dot plot.
cex	The amount by which points and symbols should be magnified relative to the default.
breaks	A numeric vector specifying the breakpoints for binning the values in x.

stacked	A logical value indicating whether the groups should be plotted in a stacked dot plot (default is TRUE).
main	A title for the plot.
showPlot	A logical value indicating whether to display the plot. Default is TRUE.

Details

Values in x are assigned to the bins defined by breaks. The binning is performed using hist.

Value

A list cointaining:

- An invisible matrix containing NAs and numeric values representing values in a bin. The number of bins is given by the number of columns of the matrix.
- The graphic.

Examples

```
# Create some data and grouping
set.seed(1)
x <- rnorm(28)
g <- rep(1:2, 14)

# Dot plot with groups and no stacking
dotPlot(x, group = g, stacked = FALSE, pch = c(19, 20), main = "Non stacked dot plot")

# Dot plot with groups and stacking
dotPlot(x, group = g, stacked = TRUE, pch = c(19, 20), main = "Stacked dot plot")
```

---

dweibull3 *dweibull3: The Weibull Distribution (3 Parameter)*

Description

Density function, distribution function, and quantile function for the Weibull distribution with a threshold parameter.

Usage

```
dweibull3(x, shape, scale, threshold)
```

Arguments

x	A numeric vector of quantiles.
shape	The shape parameter of the Weibull distribution. Default is 1.
scale	The scale parameter of the Weibull distribution. Default is 1.
threshold	The threshold (or location) parameter of the Weibull distribution. Default is 0.

Details

The Weibull distribution with the `scale` parameter alpha, `shape` parameter c, and `threshold` parameter zeta has a density function given by:

$$f(x) = \frac{c}{\alpha}\left(\frac{x-\zeta}{\alpha}\right)^{c-1}\exp\left(-\left(\frac{x-\zeta}{\alpha}\right)^{c}\right)$$

The cumulative distribution function is given by:

$$F(x) = 1 - \exp\left(-\left(\frac{x-\zeta}{\alpha}\right)^{c}\right)$$

Value

`dweibull3` returns the density, `pweibull3` returns the distribution function, and `qweibull3` returns the quantile function for the Weibull distribution with a threshold.

Examples

```
dweibull3(x = 1, scale = 1, shape = 5, threshold = 0)
temp <- pweibull3(q = 1, scale = 1, shape = 5, threshold = 0)
temp
qweibull3(p = temp, scale = 1, shape = 5, threshold = 0)
```

---

facDesign *facDesign*

Description

Generates a 2^k full factorial design.

Usage

```
facDesign(
  k = 3,
  p = 0,
  replicates = 1,
  blocks = 1,
  centerCube = 0,
  random.seed = 1234
)
```

Arguments

k	Numeric value giving the number of factors. By default k is set to ‘3‘.
p	Numeric integer between ‘0‘ and ‘7‘. p is giving the number of additional factors in the response surface design by aliasing effects. For further information see fracDesign and fracChoose. By default p is set to ‘0‘.
replicates	Numeric value giving the number of replicates per factor combination. By default replicates is set to ‘1‘.
blocks	Numeric value giving the number of blocks. By default blocks is set to ‘1‘. Blocking is only performed for k greater 2.
centerCube	Numeric value giving the number of centerpoints within the 2^k design. By default centerCube is set to ‘0‘.
random.seed	Numeric value for setting the random seed for reproducibility.

Value

The function facDesign returns an object of class facDesign.c.

See Also

fracDesign, fracChoose, rsmDesign, pbDesign, taguchiDesign

Examples

```
# Example 1
vp.full <- facDesign(k = 3)
vp.full$.response(rnorm(2^3))
vp.full$summary()

# Example 2
vp.rep <- facDesign(k = 2, replicates = 3, centerCube = 4)
vp.rep$names(c("Name 1", "Name 2"))
vp.rep$unit(c("min", "F"))
vp.rep$lows(c(20, 40, 60))
vp.rep$highs(c(40, 60, 80))
vp.rep$summary()

# Example 3
dfac <- facDesign(k = 3, centerCube = 4)
dfac$names(c('Factor 1', 'Factor 2', 'Factor 3'))
dfac$names()
dfac$lows(c(80, 120, 1))
dfac$lows()
dfac$highs(c(120, 140, 2))
dfac$highs()
dfac$summary()
```

facDesign.c *facDesign-class: Class 'facDesign'*

Description

The facDesign.c class is used to represent factorial designs, including their factors, responses, blocks, and design matrices. This class supports various experimental designs and allows for the storage and manipulation of data related to the design and analysis of factorial experiments.

Public fields

name Character string representing the name of the factorial design.

factors List of factors involved in the factorial design, including their levels and settings.

cube Data frame containing the design matrix for the cube portion of the factorial design.

star Data frame containing the design matrix for the star portion of the factorial design.

centerCube Data frame containing the center points within the cube portion of the factorial design.

centerStar Data frame containing the center points within the star portion of the factorial design.

generator List of generators used to create the fractional factorial design.

response Data frame containing the responses or outcomes measured in the design.

block Data frame specifying the block structures if the design is blocked.

blockGen Data frame specifying the block generators for the design.

runOrder Data frame specifying the order in which runs are performed.

standardOrder Data frame specifying the standard order of the runs.

desireVal List of desired values or targets for the response variables.

desirability List of desirability scores or metrics based on the desired values.

fits Data frame containing the fitted model parameters and diagnostics for the responses in the design.

Methods

Public methods:

- facDesign.c$nrow()
- facDesign.c$ncol()
- facDesign.c$print()
- facDesign.c$.clear()
- facDesign.c$names()
- facDesign.c$as.data.frame()
- facDesign.c$get()
- facDesign.c$lows()
- facDesign.c$highs()
- facDesign.c$.nfp()

- facDesign.c$identity()
- facDesign.c$summary()
- facDesign.c$.response()
- facDesign.c$effectPlot()
- facDesign.c$lm()
- facDesign.c$desires()
- facDesign.c$set.fits()
- facDesign.c$types()
- facDesign.c$unit()
- facDesign.c$.star()
- facDesign.c$.blockGen()
- facDesign.c$.block()
- facDesign.c$.centerCube()
- facDesign.c$.centerStar()
- facDesign.c$.generators()
- facDesign.c$clone()

Method `nrow()`**:** Get the number of rows Design.

Usage:

```
facDesign.c$nrow()
```

Method `ncol()`**:** Get the number of columns Design.

Usage:

```
facDesign.c$ncol()
```

Method `print()`**:** Prints a formatted representation of the factorial design object, including design matrices and responses.

Usage:

```
facDesign.c$print()
```

Method `.clear()`**:** Clears the factorial design object.

Usage:

```
facDesign.c$.clear()
```

Method `names()`**:** Get or set the names of the factors in the factorial design.

Usage:

```
facDesign.c$names(value)
```

Arguments:

`value` Character vector with new names for the factors. If missing, retrieves the current names.

Method `as.data.frame()`**:** Converts the factorial design object to a data frame.

Usage:

```
facDesign.c$as.data.frame()
```

Method `get()`**:** Retrieves elements from the factorial design object.

Usage:

```
facDesign.c$get(i, j)
```

Arguments:

`i` Row index.

`j` Column index.

Method `lows()`**:** Get or set the lower bounds of the factors in the factorial design.

Usage:

```
facDesign.c$lows(value)
```

Arguments:

`value` Numeric vector with new lower bounds. If missing, retrieves the current lower bounds.

Method `highs()`**:** Get or set the upper bounds of the factors in the factorial design.

Usage:

```
facDesign.c$highs(value)
```

Arguments:

`value` Numeric vector with new upper bounds. If missing, retrieves the current upper bounds.

Method `.nfp()`**:** Prints a summary of the factors attributes including their low, high, name, unit, and type.

Usage:

```
facDesign.c$.nfp()
```

Method `identity()`**:** Returns the factorial design object itself, used to verify or return the object.

Usage:

```
facDesign.c$identity()
```

Method `summary()`**:** Summarizes the factorial design object.

Usage:

```
facDesign.c$summary()
```

Method `.response()`**:** Get or set the response data in the factorial design object.

Usage:

```
facDesign.c$.response(value)
```

Arguments:

`value` Data frame or numeric vector with new responses. If missing, retrieves the current responses.

Method `effectPlot()`**:** Plots the effects of factors on the response variables.

Usage:

```
facDesign.c$effectPlot(
  factors,
  fun = mean,
  response = NULL,
  lty,
  xlab,
  ylab,
  main,
  ylim
)
```

Arguments:

`factors` Factors to be plotted.

`fun` Function applied to the response variables (e.g., mean).

`response` Optional; specifies which response variables to plot.

`lty` Line type for plotting.

`xlab` Label for the x-axis.

`ylab` Label for the y-axis.

`main` Main title for the plot.

`ylim` Limits for the y-axis.

Method `lm()`**:** Fits a linear model to the response data in the factorial design object.

Usage:

```
facDesign.c$lm(formula)
```

Arguments:

`formula` Formula specifying the model to be fitted.

Method `desires()`**:** Get or set the desirability values for the response variables.

Usage:

```
facDesign.c$desires(value)
```

Arguments:

`value` List of new desirability values. If missing, retrieves the current desirability values.

Method `set.fits()`**:** Set the fits for the response variables in the factorial design object.

Usage:

```
facDesign.c$set.fits(value)
```

Arguments:

`value` New fits.

Method `types()`**:** Get or set the types of designs used in the factorial design object.

Usage:

```
facDesign.c$types(value)
```

Arguments:

`value` New design types. If missing, retrieves the current types.

Method `unit()`**:** Get or set the units for the factors in the factorial design object.

Usage:

```
facDesign.c$unit(value)
```

Arguments:

`value` New units. If missing, retrieves the current units.

Method `.star()`**:** Get or set the star points in the factorial design object.

Usage:

```
facDesign.c$.star(value)
```

Arguments:

`value` New star points. If missing, retrieves the current star points.

Method `.blockGen()`**:** Get or set the block generators in the factorial design object.

Usage:

```
facDesign.c$.blockGen(value)
```

Arguments:

`value` New block generators. If missing, retrieves the current block generators.

Method `.block()`**:** Get or set the blocks in the factorial design object.

Usage:

```
facDesign.c$.block(value)
```

Arguments:

`value` New blocks. If missing, retrieves the current blocks.

Method `.centerCube()`**:** Get or set the center points in the cube portion of the factorial design.

Usage:

```
facDesign.c$.centerCube(value)
```

Arguments:

`value` New center points for the cube. If missing, retrieves the current center points.

Method `.centerStar()`**:** Get or set the center points in the star portion of the factorial design.

Usage:

```
facDesign.c$.centerStar(value)
```

Arguments:

`value` New center points for the star. If missing, retrieves the current center points.

Method `.generators()`**:** Get or set the generators for the factorial design.

Usage:

```
facDesign.c$.generators(value)
```

Arguments:

`value` New generators. If missing, retrieves the current generators.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
facDesign.c$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

mixDesign.c, taguchiDesign.c.

FitDistr *FitDistr: Maximum-likelihood Fitting of Univariate Distributions*

Description

Maximum-likelihood fitting of univariate distributions, allowing parameters to be held fixed if desired.

Usage

```
FitDistr(x, densfun, start, ...)
```

Arguments

x	A numeric vector of length at least one containing only finite values. Either a character string or a function returning a density evaluated at its first argument.
densfun	character string specifying the density function to be used for fitting the distribution. Distributions ‘"beta"‘, ‘"cauchy"‘, ‘"chi-squared"‘, ‘"exponential"‘, ‘"gamma"‘, ‘"geometric"‘, ‘"log-normal"‘, ‘"lognormal"‘, ‘"logistic"‘, ‘"negative binomial"‘, ‘"normal"‘, ‘"Poisson"‘, ‘"t"‘ and "weibull" are recognised, case being ignored.
start	A named list giving the parameters to be optimized with initial values. This can be omitted for some of the named distributions and must be for others (see Details).
...	Additional parameters, either for ‘densfun‘ or for ‘optim‘. In particular, it can be used to specify bounds via ‘lower‘ or ‘upper‘ or both. If arguments of ‘densfun‘ (or the density function corresponding to a character-string specification) are included they will be held fixed.

Details

For the Normal, log-Normal, geometric, exponential and Poisson distributions the closed-form MLEs (and exact standard errors) are used, and ‘start‘ should not be supplied.

For all other distributions, direct optimization of the log-likelihood is performed using ‘optim‘. The estimated standard errors are taken from the observed information matrix, calculated by a numerical approximation. For one-dimensional problems the Nelder-Mead method is used and for multi-dimensional problems the BFGS method, unless arguments named ‘lower‘ or ‘upper‘ are supplied (when ‘L-BFGS-B‘ is used) or ‘method‘ is supplied explicitly.

For the ‘"t"‘ named distribution the density is taken to be the location-scale family with location ‘m‘ and scale ‘s‘.

For the following named distributions, reasonable starting values will be computed if ‘start‘ is omitted or only partially specified: ‘"cauchy"‘, ‘"gamma"‘, ‘"logistic"‘, ‘"negative binomial"‘ (parametrized

by mu and size), ‘"t"‘ and ‘"weibull"‘. Note that these starting values may not be good enough if the fit is poor: in particular they are not resistant to outliers unless the fitted distribution is long-tailed.

There are ‘print‘, ‘coef‘, ‘vcov‘ and ‘logLik‘ methods for class ‘"FitDistr"‘.

Value

The function ‘FitDistr‘ returns an object of class ‘fitdistr‘, which is a list containing:

estimate	a named vector of parameter estimates.
sd	a named vector of the estimated standard errors for the parameters.
vcov	the estimated variance-covariance matrix of the parameter estimates.
loglik	the log-likelihood of the fitted model.
n	length vector.

See Also

distribution, Distr, DistrCollection.

Examples

```
set.seed(123)
x = rgamma(100, shape = 5, rate = 0.1)
FitDistr(x, "gamma")

# Now do this directly with more control.
FitDistr(x, dgamma, list(shape = 1, rate = 0.1), lower = 0.001)

set.seed(123)
x2 = rt(250, df = 9)
FitDistr(x2, "t", df = 9)

# Allow df to vary: not a very good idea!
FitDistr(x2, "t")

# Now do fixed-df fit directly with more control.
mydt = function(x, m, s, df) dt((x-m)/s, df)/s
FitDistr(x2, mydt, list(m = 0, s = 1), df = 9, lower = c(-Inf, 0))

set.seed(123)
x3 = rweibull(100, shape = 4, scale = 100)
FitDistr(x3, "weibull")
```

fracChoose — *fracChoose: Choosing a fractional or full factorial design from a table.*

Description

Designs displayed are the classic minimum abberation designs. Choosing a design is done by clicking with the mouse into the appropriate field.

Usage

```
fracChoose()
```

Value

fracChoose returns an object of class facDesign.c.

See Also

fracDesign, facDesign, rsmChoose, rsmDesign

Examples

```
fracChoose()
```

fracDesign — *fracDesign*

Description

Generates a 2^k-p fractional factorial design.

Usage

```
fracDesign(
  k = 3,
  p = 0,
  gen = NULL,
  replicates = 1,
  blocks = 1,
  centerCube = 0,
  random.seed = 1234
)
```

Arguments

k	Numeric value giving the number of factors. By default k is set to ‘3‘.
p	Numeric integer between ‘0‘ and ‘7‘. p is giving the number of additional factors in the response surface design by aliasing effects. A 2^k-p factorial design will be generated and the generators of the standard designs available in fracChoose() will be used. By default p is set to ‘0‘. Any other value will cause the function to omit the argument gen given by the user and replace it by the one out of the table of standard designs (see: fracChoose). Replicates and blocks can be set anyway!
gen	One or more defining relations for a fractional factorial design, for example: `C=AB`. By default gen is set to NULL.
replicates	Numeric value giving the number of replicates per factor combination. By default replicates is set to ‘1‘.
blocks	Numeric value giving the number of blocks. By default blocks is set to ‘1‘.
centerCube	Numeric value giving the number of center points within the 2^k design. By default centerCube is set to ‘0‘.
random.seed	Seed for randomization of the design

Value

The function fracDesign returns an object of class facDesign.c.

See Also

facDesign, fracChoose, rsmDesign, pbDesign, taguchiDesign

Examples

```
#Example 1
#Returns a 2^4-1 fractional factorial design. Factor D will be aliased with
vp.frac = fracDesign(k = 4, gen = "D=ABC")
#the three-way-interaction ABC (i.e. I = ABCD)
vp.frac$.response(rnorm(2^(4-1)))
# summary of the fractional factorial design
vp.frac$summary()

#Example 2
#Returns a full factorial design with 3 replications per factor combination and 4 center points
vp.rep = fracDesign(k = 3, replicates = 3, centerCube = 4)
#Summary of the replicated fractional factorial design
vp.rep$summary()
```

gageLin *gageLin: Function to visualize and calucalte the linearity of a gage.*

Description

Function visualize the linearity of a gage by plotting the single and mean bias in one plot and intercalate them with a straight line. Furthermore the function deliver some characteristic values of linearity studies according to MSA (Measurement System Analysis).

Usage

```
gageLin(
  object,
  conf.level = 0.95,
  ylim,
  col,
  pch,
  lty = c(1, 2),
  stats = TRUE,
  plot = TRUE
)
```

Arguments

object	An object of class MSALinearity containing the data and model for the linearity analysis. To create such an object see gageLinDesign.
conf.level	A numeric value between ‘0‘ and ‘1‘, giving the confidence intervall for the analysis. Default value: ‘0.95‘.
ylim	A numeric vector of length 2 specifying the y-axis limits for the plot. If not specified, the limits are set automatically based on the data.
col	A vector with four numeric entries. The first gives the color of the single points, the second gives the color of the points for the mean bias, the third gives the color fo the straight interpolation line and the fourth gives the color for the lines representing the confidence interval. If one of the values is missing or negative the points or lines are not plotted. col is by default ‘c(1,2,1,4)‘.
pch	A vector with two numeric or single character entries giving the symbols for the single points (1st entry) and the mean bias (2nd entry). The default vector is ‘c(20,18)‘
lty	a vector with two entries giving the line-style for the interpolating line and the confidence interval lines. For detailed information to the entries please see par. The default value for lty is ‘c(1,2)‘.
stats	Logical value. If ‘TRUE‘ (default) the function returns all calculated information.
plot	Logical value indicating whether to generate a plot of the linearity analysis. Default is TRUE.

Value

The function returns an object of class `MSALinearity` which can be used with e.g. `plot` or `summary`.

See Also

`cg`, `gageRR`, `gageLinDesign`, `MSALinearity`.

Examples

```
# Results of single runs
A=c(2.7,2.5,2.4,2.5,2.7,2.3,2.5,2.5,2.4,2.4,2.6,2.4)
B=c(5.1,3.9,4.2,5,3.8,3.9,3.9,3.9,3.9,4,4.1,3.8)
C=c(5.8,5.7,5.9,5.9,6,6.1,6,6.1,6.4,6.3,6,6.1)
D=c(7.6,7.7,7.8,7.7,7.8,7.8,7.8,7.7,7.8,7.5,7.6,7.7)
E=c(9.1,9.3,9.5,9.3,9.4,9.5,9.5,9.5,9.6,9.2,9.3,9.4)

# create Design
test=gageLinDesign(ref=c(2,4,6,8,10),n=12)
# create data.frame for results
results=data.frame(rbind(A,B,C,D,E))
# enter results in Design
test$response(results)
test$summary()

# no plot and no return
MSALin=gageLin(test,stats=FALSE,plot=FALSE)

# plot only
plot(MSALin)
MSALin$plot()

# summary
MSALin$summary()
```

`gageLinDesign` *gageLinDesign: Function to create a object of class MSALinearity.*

Description

Function generates an object that can be used with the function `gageLin`.

Usage

```
gageLinDesign(ref, n = 5)
```

Arguments

`ref`	A vector and contains the reference values for each group.
`n`	A single value and gives the amount of runs.Default value: ‘5‘..

Value

The function returns an object of class MSALinearity.

See Also

MSALinearity, gageLin.

Examples

```
# results of run A-E
A=c(2.7,2.5,2.4,2.5,2.7,2.3,2.5,2.5,2.4,2.4,2.6,2.4)
B=c(5.1,3.9,4.2,5,3.8,3.9,3.9,3.9,3.9,4,4.1,3.8)
C=c(5.8,5.7,5.9,5.9,6,6.1,6,6.1,6.4,6.3,6,6.1)
D=c(7.6,7.7,7.8,7.7,7.8,7.8,7.8,7.7,7.8,7.5,7.6,7.7)
E=c(9.1,9.3,9.5,9.3,9.4,9.5,9.5,9.5,9.6,9.2,9.3,9.4)

# create Design
test=gageLinDesign(ref=c(2,4,6,8,10),n=12)
# create data.frame for results
results=data.frame(rbind(A,B,C,D,E))
# enter results in Design
test$response(results)
```

---

gageRR — *gageRR: Gage R&R - Gage Repeatability and Reproducibility*

Description

Performs a Gage R&R analysis for an object of class gageRR.c.

Usage

```
gageRR(
  gdo,
  method = "crossed",
  sigma = 6,
  alpha = 0.25,
  tolerance = NULL,
  dig = 3,
  print = TRUE
)
```

Arguments

gdo	Needs to be an object of class gageRR.c.

method	Character string specifying the Gage R&R method. `crossed` which is the typical design for performing a Measurement Systems Analysis using Gage Repeatability and Reproducibility or `nested` which is used for destructive testing (i.e. the same part cannot be measured twice). Operators measure each a different sample of parts under the premise that the parts of each batch are alike. By default method is set to `crossed`.
sigma	Numeric value giving the number of sigmas. For sigma=6 this relates to 99.73 percent representing the full spread of a normal distribution function (i.e. pnorm(3) - pnorm(-3)). Another popular setting sigma=5.15 relates to 99 percent (i.e. pnorm(2.575) - pnorm(-2.575)). By default sigma is set to '6'.
alpha	Alpha value for discarding the interaction Operator:Part and fitting a non-interaction model. By default alpha is set to '0.25'.
tolerance	Mumeric value giving the tolerance for the measured parts. This is required to calculate the Process to Tolerance Ratio. By default tolerance is set to NULL.
dig	numeric value giving the number of significant digits for format. By default dig is set to '3'.
print	Print the summary of the perform of the Gage.

Value

The function gageRR returns an object of class gageRR.c and shows typical Gage Repeatability and Reproducibility Output including Process to Tolerance Ratios and the number of distinctive categories (i.e. ndc) the measurement system is able to discriminate with the tested setting.

See Also

gageRR.c, gageRRDesign, gageLin, cg.

Examples

```
# Create de gageRR Design
design <- gageRRDesign(Operators = 3, Parts = 10, Measurements = 3,
                       method = "crossed", sigma = 6, randomize = TRUE)
design$response(rnorm(nrow(design$X), mean = 10, sd = 2))

# Results of de Design
result <- gageRR(gdo = design, method = "crossed", sigma = 6, alpha = 0.25)
class(result)
result$plot()
```

---

gageRR.c *gageRR-class: Class 'gageRR'*

Description

R6 Class for Gage R&R (Repeatability and Reproducibility) Analysis

Public fields

`X` Data frame containing the measurement data.

`ANOVA` List containing the results of the Analysis of Variance (ANOVA) for the gage study.

`RedANOVA` List containing the results of the reduced ANOVA.

`method` Character string specifying the method used for the analysis (e.g., `` `crossed` ``, `` `nested` ``).

`Estimates` List of estimates including variance components, repeatability, and reproducibility.

`Varcomp` List of variance components.

`Sigma` Numeric value representing the standard deviation of the measurement system.

`GageName` Character string representing the name of the gage.

`GageTolerance` Numeric value indicating the tolerance of the gage.

`DateOfStudy` Character string representing the date of the gage R&R study.

`PersonResponsible` Character string indicating the person responsible for the study.

`Comments` Character string for additional comments or notes about the study.

`b` Factor levels for operator.

`a` Factor levels for part.

`y` Numeric vector or matrix containing the measurement responses.

`facNames` Character vector specifying the names of the factors (e.g., `` `Operator` ``, `` `Part` ``).

`numO` Integer representing the number of operators.

`numP` Integer representing the number of parts.

`numM` Integer representing the number of measurements per part-operator combination.

Methods

Public methods:

- `gageRR.c$new()`
- `gageRR.c$print()`
- `gageRR.c$subset()`
- `gageRR.c$summary()`
- `gageRR.c$get.response()`
- `gageRR.c$response()`
- `gageRR.c$names()`
- `gageRR.c$as.data.frame()`
- `gageRR.c$get.tolerance()`
- `gageRR.c$set.tolerance()`
- `gageRR.c$get.sigma()`
- `gageRR.c$set.sigma()`
- `gageRR.c$plot()`
- `gageRR.c$errorPlot()`
- `gageRR.c$whiskersPlot()`
- `gageRR.c$averagePlot()`

- gageRR.c$compPlot()
- gageRR.c$clone()

Method new(): Initialize the fiels of the gageRR object

Usage:

```
gageRR.c$new(
  X,
  ANOVA = NULL,
  RedANOVA = NULL,
  method = NULL,
  Estimates = NULL,
  Varcomp = NULL,
  Sigma = NULL,
  GageName = NULL,
  GageTolerance = NULL,
  DateOfStudy = NULL,
  PersonResponsible = NULL,
  Comments = NULL,
  b = NULL,
  a = NULL,
  y = NULL,
  facNames = NULL,
  numO = NULL,
  numP = NULL,
  numM = NULL
)
```

Arguments:

X Data frame containing the measurement data.

ANOVA List containing the results of the Analysis of Variance (ANOVA) for the gage study.

RedANOVA List containing the results of the reduced ANOVA.

method Character string specifying the method used for the analysis (e.g., "crossed", "nested").

Estimates List of estimates including variance components, repeatability, and reproducibility.

Varcomp List of variance components.

Sigma Numeric value representing the standard deviation of the measurement system.

GageName Character string representing the name of the gage.

GageTolerance Numeric value indicating the tolerance of the gage.

DateOfStudy Character string representing the date of the gage R&R study.

PersonResponsible Character string indicating the person responsible for the study.

Comments Character string for additional comments or notes about the study.

b Factor levels for operator.

a Factor levels for part.

y Numeric vector or matrix containing the measurement responses.

facNames Character vector specifying the names of the factors (e.g., "Operator", "Part").

numO Integer representing the number of operators.

numP Integer representing the number of parts.

`numM` Integer representing the number of measurements per part-operator combination.

Method `print()`**:** Return the data frame containing the measurement data (`X`)

Usage:

```
gageRR.c$print()
```

Method `subset()`**:** Return a subset of the data frame that containing the measurement data (`X`)

Usage:

```
gageRR.c$subset(i, j)
```

Arguments:

`i` The i-position of the row of `X`.

`j` The j-position of the column of `X`.

Method `summary()`**:** Summarize the information of the fields of the `gageRR` object.

Usage:

```
gageRR.c$summary()
```

Method `get.response()`**:** Get or get the response for a `gageRRDesign` object.

Usage:

```
gageRR.c$get.response()
```

Method `response()`**:** Set the response for a `gageRRDesign` object.

Usage:

```
gageRR.c$response(value)
```

Arguments:

`value` New response vector.

Method `names()`**:** Methods for function `names` in Package `base`.

Usage:

```
gageRR.c$names()
```

Method `as.data.frame()`**:** Methods for function `as.data.frame` in Package `base`.

Usage:

```
gageRR.c$as.data.frame()
```

Method `get.tolerance()`**:** Get the `tolerance` for an object of class `gageRR`.

Usage:

```
gageRR.c$get.tolerance()
```

Method `set.tolerance()`**:** Set the `tolerance` for an object of class `gageRR`.

Usage:

```
gageRR.c$set.tolerance(value)
```

Arguments:

`value` A data.frame or vector for the new value of tolerance.

Method `get.sigma()`**:** Get the `sigma` for an object of class `gageRR`.

Usage:

```
gageRR.c$get.sigma()
```

Method `set.sigma()`**:** Set the `sigma` for an object of class `gageRR`.

Usage:

```
gageRR.c$set.sigma(value)
```

Arguments:

`value` Valor of `sigma`

Method `plot()`**:** This function creates a customized plot using the data from the `gageRR.c` object.

Usage:

```
gageRR.c$plot(main = NULL, xlab = NULL, ylab = NULL, col, lwd, fun = mean)
```

Arguments:

`main` Character string specifying the title of the plot.

`xlab` A character string for the x-axis label.

`ylab` A character string for the y-axis label.

`col` A character string or vector specifying the color(s) to be used for the plot elements.

`lwd` A numeric value specifying the line width of plot elements

`fun` Function to use for the calculation of the interactions (e.g., `mean`, `median`). Default is `mean`.

Examples:

```
# Create gageRR-object
gdo = gageRRDesign(Operators = 3, Parts = 10, Measurements = 3, randomize = FALSE)
# Vector of responses
y = c(0.29,0.08, 0.04,-0.56,-0.47,-1.38,1.34,1.19,0.88,0.47,0.01,0.14,-0.80,
      -0.56,-1.46, 0.02,-0.20,-0.29,0.59,0.47,0.02,-0.31,-0.63,-0.46,2.26,
    1.80,1.77,-1.36,-1.68,-1.49,0.41,0.25,-0.11,-0.68,-1.22,-1.13,1.17,0.94,
     1.09,0.50,1.03,0.20,-0.92,-1.20,-1.07,-0.11, 0.22,-0.67,0.75,0.55,0.01,
    -0.20, 0.08,-0.56,1.99,2.12,1.45,-1.25,-1.62,-1.77,0.64,0.07,-0.15,-0.58,
     -0.68,-0.96,1.27,1.34,0.67,0.64,0.20,0.11,-0.84,-1.28,-1.45,-0.21,0.06,
    -0.49,0.66,0.83,0.21,-0.17,-0.34,-0.49,2.01,2.19,1.87,-1.31,-1.50,-2.16)

# Appropriate responses
gdo$response(y)
# Perform and gageRR
gdo <- gageRR(gdo)

gdo$plot()
```

Method `errorPlot()`**:** The data from an object of class `gageRR` can be analyzed by running ‘Error Charts‘ of the individual deviations from the accepted rference values. These ‘Error Charts‘ are provided by the function `errorPlot`.

Usage:

```
gageRR.c$errorPlot(main, xlab, ylab, col, pch, ylim, legend = TRUE)
```

Arguments:

`main` a main title for the plot.

`xlab` A character string for the x-axis label.

`ylab` A character string for the y-axis label.

`col` Plotting color.

`pch` An integer specifying a symbol or a single character to be used as the default in plotting points.

`ylim` The y limits of the plot.

`legend` A logical value specifying whether a legend is plotted automatically. By default legend is set to 'TRUE'.

Examples:

```
# Create gageRR-object
gdo = gageRRDesign(Operators = 3, Parts = 10, Measurements = 3, randomize = FALSE)
# Vector of responses
y = c(0.29,0.08, 0.04,-0.56,-0.47,-1.38,1.34,1.19,0.88,0.47,0.01,0.14,-0.80,
      -0.56,-1.46, 0.02,-0.20,-0.29,0.59,0.47,0.02,-0.31,-0.63,-0.46,2.26,
    1.80,1.77,-1.36,-1.68,-1.49,0.41,0.25,-0.11,-0.68,-1.22,-1.13,1.17,0.94,
     1.09,0.50,1.03,0.20,-0.92,-1.20,-1.07,-0.11, 0.22,-0.67,0.75,0.55,0.01,
    -0.20, 0.08,-0.56,1.99,2.12,1.45,-1.25,-1.62,-1.77,0.64,0.07,-0.15,-0.58,
     -0.68,-0.96,1.27,1.34,0.67,0.64,0.20,0.11,-0.84,-1.28,-1.45,-0.21,0.06,
    -0.49,0.66,0.83,0.21,-0.17,-0.34,-0.49,2.01,2.19,1.87,-1.31,-1.50,-2.16)

# Appropriate responses
gdo$response(y)
# Perform and gageRR
gdo <- gageRR(gdo)

gdo$errorPlot()
```

Method `whiskersPlot()`**:** In a Whiskers Chart, the high and low data values and the average (median) by part-by-operator are plotted to provide insight into the consistency between operators, to indicate outliers and to discover part-operator interactions. The Whiskers Chart reminds of boxplots for every part and every operator.

Usage:

```
gageRR.c$whiskersPlot(main, xlab, ylab, col, ylim, legend = TRUE)
```

Arguments:

`main` a main title for the plot.

`xlab` A character string for the x-axis label.

`ylab` A character string for the y-axis label.

`col` Plotting color.

`ylim` The y limits of the plot.

`legend` A logical value specifying whether a legend is plotted automatically. By default legend is set to 'TRUE'.

Examples:

```
# Create gageRR-object
gdo = gageRRDesign(Operators = 3, Parts = 10, Measurements = 3, randomize = FALSE)
# Vector of responses
y = c(0.29,0.08, 0.04,-0.56,-0.47,-1.38,1.34,1.19,0.88,0.47,0.01,0.14,-0.80,
      -0.56,-1.46, 0.02,-0.20,-0.29,0.59,0.47,0.02,-0.31,-0.63,-0.46,2.26,
    1.80,1.77,-1.36,-1.68,-1.49,0.41,0.25,-0.11,-0.68,-1.22,-1.13,1.17,0.94,
     1.09,0.50,1.03,0.20,-0.92,-1.20,-1.07,-0.11, 0.22,-0.67,0.75,0.55,0.01,
    -0.20, 0.08,-0.56,1.99,2.12,1.45,-1.25,-1.62,-1.77,0.64,0.07,-0.15,-0.58,
     -0.68,-0.96,1.27,1.34,0.67,0.64,0.20,0.11,-0.84,-1.28,-1.45,-0.21,0.06,
    -0.49,0.66,0.83,0.21,-0.17,-0.34,-0.49,2.01,2.19,1.87,-1.31,-1.50,-2.16)

# Appropriate responses
gdo$response(y)
# Perform and gageRR
gdo <- gageRR(gdo)

gdo$whiskersPlot()
```

Method `averagePlot()`**:** `averagePlot` creates all x-y plots of averages by size out of an object of class `gageRR`. Therfore the averages of the multiple readings by each operator on each part are plotted with the reference value or overall part averages as the index.

Usage:

```
gageRR.c$averagePlot(main, xlab, ylab, col, single = FALSE)
```

Arguments:

`main` a main title for the plot.

`xlab` A character string for the x-axis label.

`ylab` A character string for the y-axis label.

`col` Plotting color.

`single` A logical value.If ‘TRUE‘ a new graphic device will be opened for each plot. By default `single` is set to ‘FALSE‘.

Examples:

```
# Create gageRR-object
gdo = gageRRDesign(Operators = 3, Parts = 10, Measurements = 3, randomize = FALSE)
# Vector of responses
y = c(0.29,0.08, 0.04,-0.56,-0.47,-1.38,1.34,1.19,0.88,0.47,0.01,0.14,-0.80,
      -0.56,-1.46, 0.02,-0.20,-0.29,0.59,0.47,0.02,-0.31,-0.63,-0.46,2.26,
    1.80,1.77,-1.36,-1.68,-1.49,0.41,0.25,-0.11,-0.68,-1.22,-1.13,1.17,0.94,
     1.09,0.50,1.03,0.20,-0.92,-1.20,-1.07,-0.11, 0.22,-0.67,0.75,0.55,0.01,
    -0.20, 0.08,-0.56,1.99,2.12,1.45,-1.25,-1.62,-1.77,0.64,0.07,-0.15,-0.58,
     -0.68,-0.96,1.27,1.34,0.67,0.64,0.20,0.11,-0.84,-1.28,-1.45,-0.21,0.06,
    -0.49,0.66,0.83,0.21,-0.17,-0.34,-0.49,2.01,2.19,1.87,-1.31,-1.50,-2.16)

# Appropriate responses
gdo$response(y)
# Perform and gageRR
gdo <- gageRR(gdo)
```

```
gdo$averagePlot()
```

Method `compPlot()`**:** `compPlot` creates comparison x-y plots of an object of class `gageRR`. The averages of the multiple readings by each operator on each part are plotted against each other with the operators as indices. This plot compares the values obtained by one operator to those of another.

Usage:

```
gageRR.c$compPlot(main, xlab, ylab, col, cex.lab, fun = NULL)
```

Arguments:

`main` a main title for the plot.

`xlab` A character string for the x-axis label.

`ylab` A character string for the y-axis label.

`col` Plotting color.

`cex.lab` The magnification to be used for x and y labels relative to the current setting of cex.

`fun` Optional function that will be applied to the multiple readings of each part. fun should be an object of class `function` like `mean`,`median`, `sum`, etc. By default, `fun` is set to 'NULL' and all readings will be plotted.

Examples:

```
# Create gageRR-object
gdo = gageRRDesign(Operators = 3, Parts = 10, Measurements = 3, randomize = FALSE)
# Vector of responses
y = c(0.29,0.08, 0.04,-0.56,-0.47,-1.38,1.34,1.19,0.88,0.47,0.01,0.14,-0.80,
      -0.56,-1.46, 0.02,-0.20,-0.29,0.59,0.47,0.02,-0.31,-0.63,-0.46,2.26,
    1.80,1.77,-1.36,-1.68,-1.49,0.41,0.25,-0.11,-0.68,-1.22,-1.13,1.17,0.94,
     1.09,0.50,1.03,0.20,-0.92,-1.20,-1.07,-0.11, 0.22,-0.67,0.75,0.55,0.01,
    -0.20, 0.08,-0.56,1.99,2.12,1.45,-1.25,-1.62,-1.77,0.64,0.07,-0.15,-0.58,
     -0.68,-0.96,1.27,1.34,0.67,0.64,0.20,0.11,-0.84,-1.28,-1.45,-0.21,0.06,
    -0.49,0.66,0.83,0.21,-0.17,-0.34,-0.49,2.01,2.19,1.87,-1.31,-1.50,-2.16)

# Appropriate responses
gdo$response(y)
# Perform and gageRR
gdo <- gageRR(gdo)

gdo$compPlot()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
gageRR.c$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
#create gageRR-object
gdo <- gageRRDesign(Operators = 3, Parts = 10, Measurements = 3, randomize = FALSE)
```

```
#vector of responses
y <- c(0.29,0.08, 0.04,-0.56,-0.47,-1.38,1.34,1.19,0.88,0.47,0.01,0.14,-0.80,
      -0.56,-1.46, 0.02,-0.20,-0.29,0.59,0.47,0.02,-0.31,-0.63,-0.46,2.26,
      1.80,1.77,-1.36,-1.68,-1.49,0.41,0.25,-0.11,-0.68,-1.22,-1.13,1.17,0.94,
      1.09,0.50,1.03,0.20,-0.92,-1.20,-1.07,-0.11, 0.22,-0.67,0.75,0.55,0.01,
      -0.20, 0.08,-0.56,1.99,2.12,1.45,-1.25,-1.62,-1.77,0.64,0.07,-0.15,-0.58,
      -0.68,-0.96,1.27,1.34,0.67,0.64,0.20,0.11,-0.84,-1.28,-1.45,-0.21,0.06,
      -0.49,0.66,0.83,0.21,-0.17,-0.34,-0.49,2.01,2.19,1.87,-1.31,-1.50,-2.16)

#appropriate responses
gdo$response(y)
# perform and gageRR
gdo <- gageRR(gdo)

# Using the plots
gdo$plot()

## ------------------------------------------------
## Method `gageRR.c$plot`
## ------------------------------------------------

# Create gageRR-object
gdo = gageRRDesign(Operators = 3, Parts = 10, Measurements = 3, randomize = FALSE)
# Vector of responses
y = c(0.29,0.08, 0.04,-0.56,-0.47,-1.38,1.34,1.19,0.88,0.47,0.01,0.14,-0.80,
      -0.56,-1.46, 0.02,-0.20,-0.29,0.59,0.47,0.02,-0.31,-0.63,-0.46,2.26,
      1.80,1.77,-1.36,-1.68,-1.49,0.41,0.25,-0.11,-0.68,-1.22,-1.13,1.17,0.94,
      1.09,0.50,1.03,0.20,-0.92,-1.20,-1.07,-0.11, 0.22,-0.67,0.75,0.55,0.01,
      -0.20, 0.08,-0.56,1.99,2.12,1.45,-1.25,-1.62,-1.77,0.64,0.07,-0.15,-0.58,
      -0.68,-0.96,1.27,1.34,0.67,0.64,0.20,0.11,-0.84,-1.28,-1.45,-0.21,0.06,
      -0.49,0.66,0.83,0.21,-0.17,-0.34,-0.49,2.01,2.19,1.87,-1.31,-1.50,-2.16)

# Appropriate responses
gdo$response(y)
# Perform and gageRR
gdo <- gageRR(gdo)

gdo$plot()

## ------------------------------------------------
## Method `gageRR.c$errorPlot`
## ------------------------------------------------

# Create gageRR-object
gdo = gageRRDesign(Operators = 3, Parts = 10, Measurements = 3, randomize = FALSE)
# Vector of responses
y = c(0.29,0.08, 0.04,-0.56,-0.47,-1.38,1.34,1.19,0.88,0.47,0.01,0.14,-0.80,
      -0.56,-1.46, 0.02,-0.20,-0.29,0.59,0.47,0.02,-0.31,-0.63,-0.46,2.26,
      1.80,1.77,-1.36,-1.68,-1.49,0.41,0.25,-0.11,-0.68,-1.22,-1.13,1.17,0.94,
      1.09,0.50,1.03,0.20,-0.92,-1.20,-1.07,-0.11, 0.22,-0.67,0.75,0.55,0.01,
      -0.20, 0.08,-0.56,1.99,2.12,1.45,-1.25,-1.62,-1.77,0.64,0.07,-0.15,-0.58,
      -0.68,-0.96,1.27,1.34,0.67,0.64,0.20,0.11,-0.84,-1.28,-1.45,-0.21,0.06,
      -0.49,0.66,0.83,0.21,-0.17,-0.34,-0.49,2.01,2.19,1.87,-1.31,-1.50,-2.16)
```

```
# Appropriate responses
gdo$response(y)
# Perform and gageRR
gdo <- gageRR(gdo)

gdo$errorPlot()

## ------------------------------------------------
## Method `gageRR.c$whiskersPlot`
## ------------------------------------------------

# Create gageRR-object
gdo = gageRRDesign(Operators = 3, Parts = 10, Measurements = 3, randomize = FALSE)
# Vector of responses
y = c(0.29,0.08, 0.04,-0.56,-0.47,-1.38,1.34,1.19,0.88,0.47,0.01,0.14,-0.80,
      -0.56,-1.46, 0.02,-0.20,-0.29,0.59,0.47,0.02,-0.31,-0.63,-0.46,2.26,
      1.80,1.77,-1.36,-1.68,-1.49,0.41,0.25,-0.11,-0.68,-1.22,-1.13,1.17,0.94,
      1.09,0.50,1.03,0.20,-0.92,-1.20,-1.07,-0.11, 0.22,-0.67,0.75,0.55,0.01,
      -0.20, 0.08,-0.56,1.99,2.12,1.45,-1.25,-1.62,-1.77,0.64,0.07,-0.15,-0.58,
      -0.68,-0.96,1.27,1.34,0.67,0.64,0.20,0.11,-0.84,-1.28,-1.45,-0.21,0.06,
      -0.49,0.66,0.83,0.21,-0.17,-0.34,-0.49,2.01,2.19,1.87,-1.31,-1.50,-2.16)

# Appropriate responses
gdo$response(y)
# Perform and gageRR
gdo <- gageRR(gdo)

gdo$whiskersPlot()

## ------------------------------------------------
## Method `gageRR.c$averagePlot`
## ------------------------------------------------

# Create gageRR-object
gdo = gageRRDesign(Operators = 3, Parts = 10, Measurements = 3, randomize = FALSE)
# Vector of responses
y = c(0.29,0.08, 0.04,-0.56,-0.47,-1.38,1.34,1.19,0.88,0.47,0.01,0.14,-0.80,
      -0.56,-1.46, 0.02,-0.20,-0.29,0.59,0.47,0.02,-0.31,-0.63,-0.46,2.26,
      1.80,1.77,-1.36,-1.68,-1.49,0.41,0.25,-0.11,-0.68,-1.22,-1.13,1.17,0.94,
      1.09,0.50,1.03,0.20,-0.92,-1.20,-1.07,-0.11, 0.22,-0.67,0.75,0.55,0.01,
      -0.20, 0.08,-0.56,1.99,2.12,1.45,-1.25,-1.62,-1.77,0.64,0.07,-0.15,-0.58,
      -0.68,-0.96,1.27,1.34,0.67,0.64,0.20,0.11,-0.84,-1.28,-1.45,-0.21,0.06,
      -0.49,0.66,0.83,0.21,-0.17,-0.34,-0.49,2.01,2.19,1.87,-1.31,-1.50,-2.16)

# Appropriate responses
gdo$response(y)
# Perform and gageRR
gdo <- gageRR(gdo)

gdo$averagePlot()

## ------------------------------------------------
```

```
## Method `gageRR.c$compPlot`
## ------------------------------------------------

# Create gageRR-object
gdo = gageRRDesign(Operators = 3, Parts = 10, Measurements = 3, randomize = FALSE)
# Vector of responses
y = c(0.29,0.08, 0.04,-0.56,-0.47,-1.38,1.34,1.19,0.88,0.47,0.01,0.14,-0.80,
      -0.56,-1.46, 0.02,-0.20,-0.29,0.59,0.47,0.02,-0.31,-0.63,-0.46,2.26,
      1.80,1.77,-1.36,-1.68,-1.49,0.41,0.25,-0.11,-0.68,-1.22,-1.13,1.17,0.94,
      1.09,0.50,1.03,0.20,-0.92,-1.20,-1.07,-0.11, 0.22,-0.67,0.75,0.55,0.01,
      -0.20, 0.08,-0.56,1.99,2.12,1.45,-1.25,-1.62,-1.77,0.64,0.07,-0.15,-0.58,
      -0.68,-0.96,1.27,1.34,0.67,0.64,0.20,0.11,-0.84,-1.28,-1.45,-0.21,0.06,
      -0.49,0.66,0.83,0.21,-0.17,-0.34,-0.49,2.01,2.19,1.87,-1.31,-1.50,-2.16)

# Appropriate responses
gdo$response(y)
# Perform and gageRR
gdo <- gageRR(gdo)

gdo$compPlot()
```

---

gageRRDesign *gageRRDesign: Gage R&R - Gage Repeatability and Reproducibility*

Description

Function to Creates a Gage R&R design.

Usage

```
gageRRDesign(
  Operators = 3,
  Parts = 10,
  Measurements = 3,
  method = "crossed",
  sigma = 6,
  randomize = TRUE
)
```

Arguments

Operators	Numeric value giving a number or a character vector defining the Operators. By default Operators is set to ‘3‘.
Parts	A number or character vector defining the Parts. By default parts is set to ‘10‘.
Measurements	A number defining the measurements per part. By default Measurements is set to ‘3‘.

method	Character string specifying the Gage R&R method. `crossed` which is the typical design for performing a Measurement Systems Analysis using Gage Repeatability and Reproducibility or `nested` which is used for destructive testing (i.e. the same part cannot be measured twice). Operators measure each a different sample of parts under the premise that the parts of each batch are alike. By default method is set to `crossed`.
sigma	For sigma=6 this relates to 99.73 percent representing the full spread of a normal distribution function (i.e. pnorm(3) - pnorm(-3)). Another popular setting sigma=5.15 relates to 99 percent (i.e. pnorm(2.575) - pnorm(-2.575)). By default sigma is set to '6'.
randomize	Logical value. TRUE (default) randomizes the gageRR design.

Value

The function gageRRDesign returns an object of class gageRR.

See Also

gageRR.c, gageRR.

Examples

```
design <- gageRRDesign(Operators = 3, Parts = 10, Measurements = 3,
                       method = "crossed", sigma = 6, randomize = TRUE)
```

---

interactionPlot *interactionPlot*

Description

Creates an interaction plot for the factors in a factorial design to visualize the interaction effects between them.

Usage

```
interactionPlot(dfac, response = NULL, fun = mean, main, col = 1:2)
```

Arguments

dfac	An object of class facDesign.c, representing a factorial design.
response	Response variable. If the response data frame of fdo consists of more then one responses, this variable can be used to choose just one column of the response data frame. response Needs to be an object of class character with length of '1'. It needs to be the same character as the name of the response in the response data frame that should be plotted.
fun	Function to use for the calculation of the interactions (e.g., mean, median). Default is mean.

main	Character string: title of the plot.
col	Vector of colors for the plot. Single colors can be given as character strings or numeric values. Default is 1:2.

Details

interactionPlot() displays interactions for an object of class facDesign (i.e. 2^k full or 2^k-p fractional factorial design). Parts of the original interactionPlot were integrated.

Value

Return an interaction plot for the factors in a factorial design.

See Also

fracDesign, facDesign

Examples

```
# Example 1
# Create the facDesign object
dfac <- facDesign(k = 3, centerCube = 4)
dfac$names(c('Factor 1', 'Factor 2', 'Factor 3'))

# Assign performance to the factorial design
rend <- c(simProc(120,140,1), simProc(80,140,1), simProc(120,140,2),
          simProc(120,120,1), simProc(90,130,1.5), simProc(90,130,1.5),
          simProc(80,120,2), simProc(90,130,1.5), simProc(90,130,1.5),
          simProc(120,120,2), simProc(80,140,2), simProc(80,120,1))
dfac$.response(rend)

# Create an interaction plot
interactionPlot(dfac, fun = mean, col = c("purple", "red"))

# Example 2
vp <- fracDesign(k=3, replicates = 2)
y <- 4*vp$get(j=1) -7*vp$get(j=2) + 2*vp$get(j=2)*vp$get(j=1) +
     0.2*vp$get(j=3) + rnorm(16)
vp$.response(y)

interactionPlot(vp)
```

---

mixDesign *mixDesign: Mixture Designs*

Description

Function to generate simplex lattice and simplex centroid mixture designs with optional center points and axial points.

Usage

```
mixDesign(
  p,
  n = 3,
  type = "lattice",
  center = TRUE,
  axial = FALSE,
  delta,
  replicates = 1,
  lower,
  total = 1,
  randomize,
  seed = 1234
)
```

Arguments

`p`	Numerical value giving the amount of factors.
`n`	Numerical value specifying the degree (ignored if type = ‘centroid‘).
`type`	Character string giving the type of design. `type` can be ‘lattice‘ or ‘centroid‘ (referencing to the first source under the section references]. By default `type` is set to ‘lattice‘.
`center`	Logical value specifying whether (optional) center points will be added. By default ‘center‘ is set to ‘TRUE‘.
`axial`	Logical value specifying whether (optional) axial points will be added. By default ‘axial‘ is set to ‘FALSE‘.
`delta`	Numerical value giving the delta (see references) for axial runs. No default setting.
`replicates`	Vector with the number of replicates for the different design points i.e. c(center = 1, axial = 1, pureBlend = 1, BinaryBlend = 1, p-3 blend, p-2 blend, p-1 blend). By default ‘replicates‘ is set to ‘1‘.
`lower`	Numeric vector of lower-bound constraints on the component proportions (i.e. must be given in percent).
`total`	Numeric vector with • [1] the percentage of the mixture made up by the q - components (e.g. q = 3 and x1 + x2 + x3 = 0.8, total = 0.8 with 0.2 for the other factors being held constant) • [2] overall total in corresponding units (e.g. 200ml for the overall mixture)
`randomize`	Logical value. If ‘TRUE‘ the RunOrder of the mixture design will be randomized (default).
`seed`	Nmerical value giving the input for set.seed.

Value

The function `mixDesig()` returns an object of class `mixDesig()`.

Note

In this version the creation of (augmented) lattice, centroid mixture designs is fully supported. Getters and Setter methods for the mixDesign object exist just as for objects of class `facDesign` (i.e. factorial designs).

The creation of constrained component proportions is partially supported but don't rely on it. Visualization (i.e. ternary plots) for some of these designs can be done with the help of the `wirePlot3` and `contourPlot3` function.

See Also

`mixDesign.c`, `facDesign.c`, `facDesign`, `fracDesign`, `rsmDesign`, `wirePlot3`, `contourPlot3`.

Examples

```
# Example usage of mixDesign
mdo <- mixDesign(3, 2, center = FALSE, axial = FALSE, randomize = FALSE, replicates = c(1, 1, 2, 3))

mdo$names(c("polyethylene", "polystyrene", "polypropylene"))
elongation <- c(11.0, 12.4, 15.0, 14.8, 16.1, 17.7,
                16.4, 16.6, 8.8, 10.0, 10.0, 9.7,
                11.8, 16.8, 16.0)
mdo$.response(elongation)

mdo$units()
mdo$summary()
```

`mixDesign.c` *mixDesign-class: Class 'mixDesign'*

Description

mixDesign class for simplex lattice and simplex centroid mixture designs with optional center points and augmented points.

Public fields

`name` Character string representing the name of the design.

`factors` List of factors involved in the mixture design, including their levels and settings.

`total` Numeric value representing the total number of runs in the design.

`lower` Numeric vector representing the lower bounds of the factors in the design.

`design` Data frame containing the design matrix for the mixture design.

`designType` Character string specifying the type of design (e.g., "simplex-lattice", "simplex-centroid").

`pseudo` Data frame containing pseudo-experimental runs if applicable.

`response` Data frame containing the responses or outcomes measured in the design.

`Type` Data frame specifying the type of design used (e.g., "factorial", "response surface").

`block` Data frame specifying block structures if the design is blocked.

`runOrder` Data frame specifying the order in which runs are performed.

`standardOrder` Data frame specifying the standard order of the runs.

`desireVal` List of desired values or targets for the response variables.

`desirability` List of desirability scores or metrics based on the desired values.

`fits` Data frame containing the fitted model parameters and diagnostics.

Methods

Public methods:

- `mixDesign.c$.factors()`
- `mixDesign.c$names()`
- `mixDesign.c$as.data.frame()`
- `mixDesign.c$print()`
- `mixDesign.c$.response()`
- `mixDesign.c$.nfp()`
- `mixDesign.c$summary()`
- `mixDesign.c$units()`
- `mixDesign.c$lows()`
- `mixDesign.c$highs()`
- `mixDesign.c$clone()`

Method `.factors()`**:** Get and set the `factors` in an object of class `mixDesign`

Usage:

```
mixDesign.c$.factors(value)
```

Arguments:

`value` New factors, If missing value get the `factors`.

Method `names()`**:** Get and set the `names` in an object of class `mixDesign`.

Usage:

```
mixDesign.c$names(value)
```

Arguments:

`value` New names, If missing value get the `names`.

Method `as.data.frame()`**:** Methods for function `as.data.frame` in Package `base`.

Usage:

```
mixDesign.c$as.data.frame()
```

Method `print()`**:** Methods for function `print` in Package `base`.

Usage:

```
mixDesign.c$print()
```

Method `.response()`**:** Get and set the the `response` in an object of class `mixDesign`.

Usage:

```
mixDesign.c$.response(value)
```

Arguments:

`value` New response, If missing value get the `response`.

Method `.nfp()`**:** Prints a summary of the factors attributes including their low, high, name, unit, and type.

Usage:

```
mixDesign.c$.nfp()
```

Method `summary()`**:** Methods for function `summary` in Package `base`.

Usage:

```
mixDesign.c$summary()
```

Method `units()`**:** Get and set the `units` for the factors in an object of class `mixDesign`.

Usage:

```
mixDesign.c$units(value)
```

Arguments:

`value` New units, If missing value get the `units`.

Method `lows()`**:** Get and set the `lows` for the factors in an object of class `mixDesign`.

Usage:

```
mixDesign.c$lows(value)
```

Arguments:

`value` New lows, If missing value get the `lows`.

Method `highs()`**:** Get and set the `highs` for the factors in an object of class `mixDesign`.

Usage:

```
mixDesign.c$highs(value)
```

Arguments:

`value` New highs, If missing value get the `highs`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
mixDesign.c$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`mixDesign`, `contourPlot3`, `wirePlot3`

MSALinearity *MSALinearity-class: Class 'MSALinearity'*

Description

R6 class for performing Measurement System Analysis (MSA) Linearity studies.

Public fields

`X` A data frame containing the independent variable(s) used in the linearity study.

`Y` A data frame containing the dependent variable(s) or responses measured in the linearity study.

`model` The linear model object resulting from the linearity analysis.

`conf.level` A numeric value specifying the confidence level for the linearity analysis. This should be between 0 and 1 (e.g., 0.95 for a 95% confidence level).

`Linearity` A list or data frame containing the results of the linearity study, including the linearity value and associated statistics.

`GageName` A character string specifying the name of the gage or measurement system under analysis.

`GageTolerance` A numeric value specifying the tolerance of the gage or measurement system.

`DateOfStudy` A character string or Date object indicating the date when the linearity study was conducted.

`PersonResponsible` A character string specifying the name of the person responsible for the linearity study.

`Comments` A character string for additional comments or notes about the linearity study.

`facNames` A character vector specifying the names of the factors involved in the study, if any.

Methods

Public methods:

- `MSALinearity$response()`
- `MSALinearity$summary()`
- `MSALinearity$plot()`
- `MSALinearity$print()`
- `MSALinearity$as.data.frame()`
- `MSALinearity$clone()`

Method `response()`**:** Get and set the the `response` in an object of class `MSALinearity`.

Usage:

```
MSALinearity$response(value)
```

Arguments:

`value` New response, If missing value get the `response`.

Method `summary()`**:** Methods for function `summary` in Package `base`.

Usage:

```
MSALinearity$summary()
```

Method `plot()`**:** Plots the measurement system, including individual biases, mean bias, and a regression line with confidence intervals.

Usage:

```
MSALinearity$plot(ylim, col, pch, lty = c(1, 2))
```

Arguments:

`ylim` A numeric vector specifying the limits for the y-axis. If not provided, the limits are automatically calculated based on data.

`col` A vector specifying the colors to be used for different plot elements.

`pch` A numeric vector specifying the plotting characters (symbols) for individual data points and mean bias points.

`lty` A numeric vector specifying the line types for the regression line and its confidence intervals. The default is `c(1, 2)`.

Method `print()`**:** Methods for function `print` in Package `base`.

Usage:

```
MSALinearity$print()
```

Method `as.data.frame()`**:** Return a data frame with the information of the object `MSALinearity`.

Usage:

```
MSALinearity$as.data.frame()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
MSALinearity$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`mvPlot` *mvPlot: Function to create a multi-variable plot*

Description

Creates a plot for visualizing the relationships between a response variable and multiple factors.

Usage

```
mvPlot(
  response,
  fac1,
  fac2,
  fac3,
  fac4,
  sort = TRUE,
  col,
  pch,
  labels = FALSE,
  quantile = TRUE,
  FUN = NA
)
```

Arguments

response	The values of the response in a vector.response must be declared.
fac1	Vector providing factor 1 as shown in the example.fac1 must be declared.
fac2	Vector providing factor 1 as shown in the example.fac2 must be declared.
fac3	Optional vector providing factor 3 as shown in the example.
fac4	Optional vector providing factor 4 as shown in the example.
sort	Logical value indicating whether the sequence of the factors given by fac1 - fac4 should be reordered to minimize the space needed to visualize the Multi-Vari-Chart. By default sort is set to 'TRUE'.
col	Graphical parameter. Vector containing numerical values or character strings giving the colors for the different factors. By default col starts with the value '3' and is continued as needed.
pch	Graphical parameter. Vector containing numerical values or single characters giving plotting points for the different factors. See points for possible values and their interpretation. Note that only integers and single-character strings can be set as a graphics parameter (and not NA nor NULL). By default pch starts with the value '1' and is continued as needed.
labels	Logical value indicating whether the single points should be labels with the row-number of the data.frame invisibly returned by the function mvPlot. By default labels is set to 'FALSE'.
quantile	A logical value indicating whether the quanitiles (0.00135, 0.5 & 0.99865) should be visualized for the single groups. By default quantile is set to 'TRUE'.
FUN	An optional function to be used for calculation of response for unique settings of the factors e.g. the mean. By default FUN is set to 'NA' and therfore omitted.

Value

mvPlot returns an invisible list cointaining: a data.frame in which all plotted points are listed and the final plot. The option labels can be used to plot the row-numbers at the single points and to ease the identification.

Examples

```
#Example I
examp1 = expand.grid(c("Engine1","Engine2","Engine3"),c(10,20,30,40))
examp1 = as.data.frame(rbind(examp1, examp1, examp1))
examp1 = cbind(examp1, rnorm(36, 1, 0.02))
names(examp1) = c("factor1", "factor2", "response")
mvPlot(response = examp1[,3], fac1 = examp1[,2],fac2 = examp1[,1],sort=FALSE,FUN=mean)
```

`normalPlot` *normalPlot: Normal plot*

Description

Creates a normal probability plot for the effects in a `facDesign.c` object.

Usage

```
normalPlot(
  dfac,
  response = NULL,
  main,
  ylim,
  xlim,
  xlab,
  ylab,
  pch,
  col,
  border = "red"
)
```

Arguments

`dfac`	An object of class `facDesign.c`.
`response`	Response variable. If the response data frame of fdo consists of more then one responses, this variable can be used to choose just one column of the `response` data frame. response needs to be an object of class character with length of ‘1‘. It needs to be the same character as the name of the response in the response data frame that should be plotted. By default `respons‘` is set to `NULL`.
`main`	Character string specifying the main title of the plot.
`ylim`	Graphical parameter. The y limits of the plot.
`xlim`	Graphical parameter. The x limits (x1, x2) of the plot. Note that x1 > x2 is allowed and leads to a ‘reversed axis‘.
`xlab`	Character string specifying the label for the x-axis.
`ylab`	Character string specifying the label for the y-axis.

pch	Graphical parameter. Vector containing numerical values or single characters giving plotting points for the different factors. Accepts values from 0 to 25, each corresponding to a specific shape in `ggplot2` (e.g., 0: square, 1: circle, 2: triangle point up, 3: plus, 4: cross).
col	Graphical parameter. Single numerical value or character string giving the color for the points (e.g., 1: black, 2: red, 3: green).
border	Graphical parameter. Single numerical value or character string giving the color of the border line.

Details

If the given `facDesign.c` object `fdo` contains replicates this function will deliver a normal plot i.e.: effects divided by the standard deviation (t-value) will be plotted against an appropriate probability scaling (see: ‘ppoints‘). If the given `facDesign.c` object `fdo` contains no replications the standard error can not be calculated. In that case the function will deliver an effect plot. i.e.: the effects will be plotted against an appropriate probability scaling. (see: ‘ppoints‘).

Value

The function `normalPlot` returns an invisible list containing:

effects	a list of effects for each response in the `facDesign.c` object.
plot	The generated normal plot.

See Also

`facDesign`, `paretoPlot`, `interactionPlot`

Examples

```
# Example 1: Create a normal probability plot for a full factorial design
dfac <- facDesign(k = 3, centerCube = 4)
dfac$names(c('Factor 1', 'Factor 2', 'Factor 3'))

# Assign performance to the factorial design
rend <- c(simProc(120,140,1), simProc(80,140,1), simProc(120,140,2),
          simProc(120,120,1), simProc(90,130,1.5), simProc(90,130,1.5),
          simProc(80,120,2), simProc(90,130,1.5), simProc(90,130,1.5),
          simProc(120,120,2), simProc(80,140,2), simProc(80,120,1))
dfac$.response(rend)

normalPlot(dfac)

# Example 2: Create a normal probability plot with custom colors and symbols
normalPlot(dfac, col = "blue", pch = 4)
```

oaChoose *oaChoose: Taguchi Designs*

Description

Shows a matrix of possible taguchi designs.

Usage

```
oaChoose(factors1, factors2, level1, level2, ia)
```

Arguments

factors1	Number of factors on level1.
factors2	Number of factors on level2.
level1	Number of levels on level1.
level2	Number of levels on level2.
ia	Number of interactions.

Details

oaChoose returns possible taguchi designs. Specifying the number of factor1 factors with level1 levels (factors1 = 2, level1 = 3 means 2 factors with 3 factor levels) and factor2 factors with level2 levels and desired interactions one or more taguchi designs are suggested. If all parameters are set to '0', a matrix of possible taguchi designs is shown.

Value

oaChoose returns an object of class taguchiDesign.

See Also

- facDesign: for 2^k factorial designs.
- rsmDesign: for response surface designs.
- fracDesign: for fractional factorial design.
- gageRRDesign: for gage designs.

Examples

```
oaChoose()
```

optimum *optimum: Optimal factor settings*

Description

This function calculates the optimal factor settings based on defined desirabilities and constraints. It supports two approaches: (I) evaluating all possible factor settings via a grid search and (II) using optimization methods such as `optim` or `gosolnp` from the Rsolnp package. Using `optim` initial values for the factors to be optimized over can be set via start. The optimality of the solution depends critically on the starting parameters which is why it is recommended to use type=`gosolnp` although calculation takes a while.

Usage

```
optimum(fdo, constraints, steps = 25, type = "grid", start)
```

Arguments

fdo	An object of class facDesign.c with fits and desires set.
constraints	A list specifying the constraints for the factors, e.g., list(A = c(-2,1), B = c(0, 0.8)).
steps	Number of grid points per factor if type = `grid`. Default is '25'.
type	The type of search to perform. Supported values are `grid`, `optim`, and `gosolnp`. See Details for more information.
start	A numeric vector providing the initial values for the factors when using type = `optim`.

Details

The function allows you to optimize the factor settings either by evaluating a grid of possible settings (type = `grid`) or by using optimization algorithms (type = `optim` or `gosolnp`). The choice of optimization method may significantly affect the result, especially for desirability functions that lack continuous first derivatives. When using type = `optim`, it is advisable to provide start values to avoid local optima. The `gosolnp` method is recommended for its robustness, although it may be computationally intensive.

Value

Return an object of class desOpt.

See Also

overall, desirability,

Examples

```
#Example 1: Simultaneous Optimization of Several Response Variables
#Define the response surface design as given in the paper and sort via Standard Order
fdo = rsmDesign(k = 3, alpha = 1.633, cc = 0, cs = 6)
fdo = randomize(fdo, so = TRUE)
#Attaching the 4 responses
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)
fdo$.response(data.frame(y1, y2, y3, y4)[c(5,2,3,8,1,6,7,4,9:20),])
#Setting names and real values of the factors
fdo$names(c("silica", "silan", "sulfur"))
fdo$highs(c(1.7, 60, 2.8))
fdo$lows(c(0.7, 40, 1.8))
#Setting the desires
fdo$desires(desirability(y1, 120, 170, scale = c(1,1), target = "max"))
fdo$desires(desirability(y2, 1000, 1300, target = "max"))
fdo$desires(desirability(y3, 400, 600, target = 500))
fdo$desires(desirability(y4, 60, 75, target = 67.5))
#Setting the fits
fdo$set.fits(fdo$lm(y1 ~ A + B + C + A:B + A:C + B:C + I(A^2) + I(B^2) + I(C^2)))
fdo$set.fits(fdo$lm(y2 ~ A + B + C + A:B + A:C + B:C + I(A^2) + I(B^2) + I(C^2)))
fdo$set.fits(fdo$lm(y3 ~ A + B + C + A:B + A:C + B:C + I(A^2) + I(B^2) + I(C^2)))
fdo$set.fits(fdo$lm(y4 ~ A + B + C + A:B + A:C + B:C + I(A^2) + I(B^2) + I(C^2)))
#Calculate the best factor settings using type = "optim"
optimum(fdo, type = "optim")
#Calculate the best factor settings using type = "grid"
optimum(fdo, type = "grid")
```

`overall` *overall: Overall Desirability.*

Description

This function calculates the desirability for each response as well as the overall desirability. The resulting `data.frame` can be used to plot the overall desirability as well as the desirabilities for each response. This function is designed to visualize the desirability approach for multiple response optimization.

Usage

```
overall(fdo, steps = 20, constraints, ...)
```

Arguments

`fdo`	An object of class `facDesign.c` containing `fits` and `desires`.
`steps`	A numeric value indicating the number of points per factor to be evaluated, which also specifies the grid size. Default is '20'.
`constraints`	A list of constraints for the factors in coded values, such as `list(A > 0.5, B < 0.2)`.
`...`	Further arguments passed to other methods.

Value

A `data.frame` with a column for each factor, the desirability for each response, and a column for the overall desirability.

See Also

`facDesign`, `rsmDesign`, `desirability`.

Examples

```
#Example 1: Arbitrary example with random data
rsdo = rsmDesign(k = 2, blocks = 2, alpha = "both")
rsdo$.response(data.frame(y = rnorm(rsdo$nrow()), y2 = rnorm(rsdo$nrow())))
rsdo$set.fits(rsdo$lm(y ~ A*B + I(A^2) + I(B^2)))
rsdo$set.fits(rsdo$lm(y2 ~ A*B + I(A^2) + I(B^2)))
rsdo$desires(desirability(y, -1, 2, scale = c(1, 1), target = "max"))
rsdo$desires(desirability(y2, -1, 0, scale = c(1, 1), target = "min"))
dVals = overall(rsdo, steps = 10, constraints = list(A = c(-0.5,1), B = c(0, 1)))
```

`paretoChart` *paretoChart: Pareto Chart*

Description

Function to create a Pareto chart, displaying the relative frequency of categories.

Usage

```
paretoChart(
  x,
  weight,
  main,
  col,
  border,
  xlab,
  ylab = "Frequency",
  percentVec,
  showTable = TRUE,
  showPlot = TRUE
)
```

Arguments

x	A vector of qualitative values.
weight	A numeric vector of weights corresponding to each category in x.
main	A character string for the main title of the plot.
col	A numerical value or character string defining the fill-color of the bars.
border	A numerical value or character string defining the border-color of the bars.
xlab	A character string for the x-axis label.
ylab	A character string for the y-axis label. By default, ylab is set to `Frequency`.
percentVec	A numerical vector giving the position and values of tick marks for percentage axis.
showTable	Logical value indicating whether to display a table of frequencies. By default, showTable is set to TRUE.
showPlot	Logical value indicating whether to display the Pareto chart. By default, showPlot is set to TRUE.

Value

paretoChart returns a Pareto chart along with a frequency table if showTable is TRUE. Additionally, the function returns an invisible list containing:

plot	The generated Pareto chart.
table	A data.frame with the frequencies and percentages of the categories.

Examples

```
# Example 1: Creating a Pareto chart for defect types
defects1 <- c(rep("E", 62), rep("B", 15), rep("F", 3), rep("A", 10),
              rep("C", 20), rep("D", 10))
paretoChart(defects1)

# Example 2: Creating a Pareto chart with weighted frequencies
```

```
defects2 <- c("E", "B", "F", "A", "C", "D")
frequencies <- c(62, 15, 3, 10, 20, 10)
weights <- c(1.5, 2, 0.5, 1, 1.2, 1.8)
names(weights) <- defects2  # Assign names to the weights vector

paretoChart(defects2, weight = frequencies * weights)
```

---

paretoPlot *paretoPlot*

Description

Display standardized effects and interactions of a facDesign.c object in a pareto plot.

Usage

```
paretoPlot(
  dfac,
  abs = TRUE,
  decreasing = TRUE,
  alpha = 0.05,
  response = NULL,
  ylim,
  xlab,
  ylab,
  main,
  p.col,
  legend_left = TRUE
)
```

Arguments

dfac	An object of class facDesign.
abs	Logical. If TRUE, absolute effects and interactions are displayed. Default is TRUE.
decreasing	Logical. If TRUE, effects and interactions are sorted decreasing. Default is TRUE.
alpha	The significance level used to calculate the critical value
response	Response variable. If the response data frame of fdo consists of more then one responses, this variable can be used to choose just one column of the response data frame. response needs to be an object of class character with length of '1'. It needs to be the same character as the name of the response in the response data frame that should be plotted. By default response is set to NULL.
ylim	Numeric vector of length 2: limits for the y-axis. If missing, the limits are set automatically.
xlab	Character string: label for the x-axis.

ylab	Character string: label for the y-axis.
main	Character string: title of the plot.
p.col	Character string specifying the color palette to use for the plot. Must be one of the following values from the RColorBrewer package: • `Set1` • `Set2` • `Set3` • `Pastel1` • `Pastel2` • `Paired` • `Dark2` • `Accent`
legend_left	Logical value indicating whether to place the legend on the left side of the plot. Default is TRUE.

Details

paretoPlot displays a pareto plot of effects and interactions for an object of class facDesign (i.e. 2^k full or 2^k-p fractional factorial design). For a given significance level alpha, a critical value is calculated and added to the plot. Standardization is achieved by dividing estimates with their standard error. For unreplicated fractional factorial designs a Lenth Plot is generated.

Value

The function paretoPlot returns an invisible list containing:

effects	a list of effects for each response in the facDesign.c object
plot	The generated PP plot.

See Also

fracDesign, facDesign

Examples

```
# Create the facDesign object
dfac <- facDesign(k = 3, centerCube = 4)
dfac$names(c('Factor 1', 'Factor 2', 'Factor 3'))

# Assign performance to the factorial design
rend <- c(simProc(120,140,1), simProc(80,140,1), simProc(120,140,2),
          simProc(120,120,1), simProc(90,130,1.5), simProc(90,130,1.5),
          simProc(80,120,2), simProc(90,130,1.5), simProc(90,130,1.5),
          simProc(120,120,2), simProc(80,140,2), simProc(80,120,1))
dfac$.response(rend)

paretoPlot(dfac)
paretoPlot(dfac, decreasing = TRUE, abs = FALSE, p.col = "Pastel1")
```

pbDesign *pbDesign: Plackett-Burman Designs*

Description

Function to create a Plackett-Burman design.

Usage

```
pbDesign(n, k, randomize = TRUE, replicates = 1)
```

Arguments

n	Integer value giving the number of trials.
k	Integer value giving the number of factors.
randomize	A logical value (TRUE/FALSE) that specifies whether to randomize the RunOrder of the design. By default, randomize is set to TRUE.
replicates	An integer specifying the number of replicates for each run in the design.

Value

A pbDesign returns an object of class pbDesign.

Note

This function creates Placket-Burman Designs down to n=26. Bigger Designs are not implemented because of lack in practicability. For the creation either the number of factors or the number of trials can be denoted. Wrong combinations will lead to an error message. Originally Placket-Burman-Design are applicable for number of trials divisible by 4. If n is not divisble by 4 this function will take the next larger Placket-Burman Design and truncate the last rows and columns.

See Also

- facDesign: for 2^k factorial designs.
- rsmDesign: for response surface designs.
- fracDesign: for fractional factorial design.
- gageRRDesign: for gage designs.

Examples

```
pbdo<- pbDesign(n=5)
pbdo$summary()
```

pbDesign.c *pbDesign*

Description

An R6 class representing a Plackett-Burman design.

Public fields

`name` A character string specifying the name of the design. Default is `NULL`.

`factors` A list of factors included in the Taguchi design. Each factor is typically an instance of the `pbFactor` class.

`design` A `data.frame` representing the design matrix of the experiment. This includes the levels of each factor for every run of the experiment. Default is an empty `data.frame`.

`designType` A character string specifying the type of Taguchi design used. Default is `NULL`.

`replic` A `data.frame` containing the replication information for the design. Default is an empty `data.frame`.

`response` A `data.frame` storing the response values collected from the experiment. Default is an empty `data.frame`.

`Type` A `data.frame` specifying the type of responses or factors involved in the design. Default is an empty `data.frame`.

`block` A `data.frame` indicating any blocking factors used in the design. Default is an empty `data.frame`.

`runOrder` A `data.frame` detailing the order in which the experimental runs were conducted. Default is an empty `data.frame`.

`standardOrder` A `data.frame` detailing the standard order of the experimental runs. Default is an empty `data.frame`.

`desireVal` A list storing desired values for responses in the experiment. Default is an empty list.

`desirability` A list storing desirability functions used to evaluate the outcomes of the experiment. Default is an empty list.

`fits` A `data.frame` containing model fits or other statistical summaries from the analysis of the experimental data. Default is an empty `data.frame`.

Methods

Public methods:

- `pbDesign.c$values()`
- `pbDesign.c$units()`
- `pbDesign.c$.factors()`
- `pbDesign.c$names()`
- `pbDesign.c$as.data.frame()`
- `pbDesign.c$print()`

- pbDesign.c$.response()
- pbDesign.c$.nfp()
- pbDesign.c$summary()
- pbDesign.c$clone()

Method `values()`**:** Get and set the `values` for an object of class `pbDesign`.

Usage:

```
pbDesign.c$values(value)
```

Arguments:

`value` New value, If missing value get the `values`.

Method `units()`**:** Get and set the `units` for an object of class `pbDesign`.

Usage:

```
pbDesign.c$units(value)
```

Arguments:

`value` New units, If missing value get the `units`.

Method `.factors()`**:** Get and set the `factors` in an object of class `pbDesign`.

Usage:

```
pbDesign.c$.factors(value)
```

Arguments:

`value` New factors, If missing value get the `factors`.

Method `names()`**:** Get and set the `names` in an object of class `pbDesign`.

Usage:

```
pbDesign.c$names(value)
```

Arguments:

`value` New names, If missing value get the `names`.

Method `as.data.frame()`**:** Return a data frame with the information of the object `pbDesign`.

Usage:

```
pbDesign.c$as.data.frame()
```

Method `print()`**:** Methods for function `print` in Package `base`.

Usage:

```
pbDesign.c$print()
```

Method `.response()`**:** Get and set the the `response` in an object of class `pbDesign`.

Usage:

```
pbDesign.c$.response(value)
```

Arguments:

`value` New response, If missing value get the `response`.

Method `.nfp()`**:** Prints a summary of the factors attributes including their low, high, name, unit, and type.

Usage:

```
pbDesign.c$.nfp()
```

Method `summary()`**:** Methods for function `summary` in Package `base`.

Usage:

```
pbDesign.c$summary()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
pbDesign.c$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`pbFactor` *pbFactor*

Description

An R6 class representing a factor in a Plackett-Burman design.

Public fields

`values` A vector containing the levels or values associated with the factor. Default is `NA`.

`name` A character string specifying the name of the factor. Default is an empty string ``.

`unit` A character string specifying the unit of measurement for the factor. Default is an empty string ``.

`type` A character string specifying the type of the factor, which can be either `numeric` or `categorical`. Default is `numeric`.

Methods

Public methods:

- `pbFactor$attributes()`
- `pbFactor$.values()`
- `pbFactor$.unit()`
- `pbFactor$names()`
- `pbFactor$clone()`

Method `attributes()`**:** Get the attributes of the factor.

Usage:

```
pbFactor$attributes()
```

Method `.values()`**:** Get and set the `values` for the factors in an object of class `pbFactor`.

Usage:

```
pbFactor$.values(value)
```

Arguments:

`value` New values, If missing value get the `values`.

Method `.unit()`**:** Get and set the `units` for the factors in an object of class `pbFactor`.

Usage:

```
pbFactor$.unit(value)
```

Arguments:

`value` New unit, If missing value get the `units`.

Method `names()`**:** Get and set the `names` in an object of class `pbFactor`.

Usage:

```
pbFactor$names(value)
```

Arguments:

`value` New names, If missing value get the `names`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
pbFactor$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`pcr` *pcr: Process Capability Indices*

Description

Calculates the process capability `cp`, `cpk`, `cpkL` (onesided) and `cpkU` (onesided) for a given dataset and distribution. A histogram with a density curve is displayed along with the specification limits and a Quantile-Quantile Plot for the specified distribution. Lower-, upper and total fraction of nonconforming entities are calculated. Box-Cox Transformations are supported as well as the calculation of Anderson Darling Test Statistics.

Usage

```
pcr(
  x,
  distribution = "normal",
  lsl,
  usl,
  target,
```

```
  boxcox = FALSE,
  lambda = c(-5, 5),
  main,
  xlim,
  grouping = NULL,
  std.dev = NULL,
  conf.level = 0.9973002,
  bounds.lty = 3,
  bounds.col = "red",
  col.fill = "lightblue",
  col.border = "black",
  col.curve = "red",
  plot = TRUE,
  ADtest = TRUE
)
```

Arguments

`x`	Numeric vector containing the values for which the process capability should be calculated.
`distribution`	Character string specifying the distribution of `x`. The function `cp` will accept the following character strings for `distribution`: • `` `normal` `` • `` `log-normal` `` • `` `exponential` `` • `` `logistic` `` • `` `gamma` `` • `` `weibull` `` • `` `cauchy` `` • `` `gamma3` `` • `` `weibull3` `` • `` `lognormal3` `` • `` `beta` `` • `` `f` `` • `` `geometric` `` • `` `poisson` `` • `` `negative-binomial` `` By default `distribution` is set to `` `normal` ``.
`lsl`	A numeric value specifying the lower specification limit.
`usl`	A numeric value specifying the upper specification limit.
`target`	(Optional) numeric value giving the target value.
`boxcox`	Logical value specifying whether a Box-Cox transformation should be performed or not. By default `boxcox` is set to `FALSE`.
`lambda`	(Optional) lambda for the transformation, default is to have the function estimate lambda.

main	A character string specifying the main title of the plot.
xlim	A numeric vector of length 2 specifying the x-axis limits for the plot.
grouping	(Optional) If grouping is given the standard deviation is calculated as mean standard deviation of the specified subgroups corrected by the factor c4 and expected fraction of nonconforming is calculated using this standard deviation.
std.dev	An optional numeric value specifying the historical standard deviation (only provided for normal distribution). If NULL, the standard deviation is calculated from the data.
conf.level	Numeric value between 0 and 1 giving the confidence interval. By default conf.level is 0.9973 (99.73%) which is the reference interval bounded by the 99.865% and 0.135% quantile.
bounds.lty	graphical parameter. For further details see ppPlot or qqPlot.
bounds.col	A character string specifying the color of the capability bounds. Default is "red".
col.fill	A character string specifying the fill color for the histogram plot. Default is "lightblue".
col.border	A character string specifying the border color for the histogram plot. Default is "black".
col.curve	A character string specifying the color of the fitted distribution curve. Default is "red".
plot	A logical value indicating whether to generate a plot. Default is TRUE.
ADtest	A logical value indicating whether to print the Anderson-Darling. Default is TRUE.

Details

Distribution fitting is delegated to the function FitDistr from this package, as well as the calculation of lambda for the Box-Cox Transformation. p-values for the Anderson-Darling Test are reported for the most important distributions.

The process capability indices are calculated as follows:

- **cpk**: minimum of cpK and cpL.
- **pt**: total fraction nonconforming.
- **pu**: upper fraction nonconforming.
- **pl**: lower fraction nonconforming.
- **cp**: process capability index.
- **cpkL**: lower process capability index.
- **cpkU**: upper process capability index.
- **cpk**: minimum process capability index.

For a Box-Cox transformation, a data vector with positive values is needed to estimate an optimal value of lambda for the Box-Cox power transformation of the values. The Box-Cox power transformation is used to bring the distribution of the data vector closer to normality. Estimation of the optimal lambda is delegated to the function boxcox from the MASS package. The Box-Cox transformation has the form $y(\lambda) = \frac{y^\lambda - 1}{\lambda}$ for $\lambda \neq 0$, and $y(\lambda) = \log(y)$ for $\lambda = 0$. The function boxcox

computes the profile log-likelihoods for a range of values of the parameter lambda. The function `boxcox.lambda` returns the value of lambda with the maximum profile log-likelihood.

In case no specification limits are given, `lsl` and `usl` are calculated to support a process capability index of 1.

Value

The function returns a list with the following components:

The function `pcr` returns a list with `lambda`, `cp`, `cpl`, `cpu`, `ppt`, `ppl`, `ppu`, `A`, `usl`, `lsl`, `target`, `asTest`, `plot`.

Examples

```
set.seed(1234)
data <- rnorm(20, mean = 20)
pcr(data, "normal", lsl = 17, usl = 23)

set.seed(1234)
weib <- rweibull(20, shape = 2, scale = 8)
pcr(weib, "weibull", usl = 20)
```

`pgamma3` *pgamma3: The gamma Distribution (3 Parameter)*

Description

Density function, distribution function, and quantile function for the Gamma distribution.

Usage

```
pgamma3(q, shape, scale, threshold)
```

Arguments

`q`	A numeric vector of quantiles.
`shape`	The shape parameter, default is `1`.
`scale`	The scale parameter, default is `1`.
`threshold`	The threshold parameter, default is `0`.

Details

The Gamma distribution with `scale` parameter alpha, `shape` parameter c, and `threshold` parameter zeta has a density given by:

$$f(x) = \frac{c}{\alpha}\left(\frac{x-\zeta}{\alpha}\right)^{c-1}\exp\left(-\left(\frac{x-\zeta}{\alpha}\right)^{c}\right)$$

The cumulative distribution function is given by:

$$F(x) = 1 - \exp\left(-\left(\frac{x-\zeta}{\alpha}\right)^c\right)$$

Value

dgamma3 gives the density, pgamma3 gives the distribution function, and qgamma3 gives the quantile function.

Examples

```
dgamma3(x = 1, scale = 1, shape = 5, threshold = 0)
temp <- pgamma3(q = 1, scale = 1, shape = 5, threshold = 0)
temp
qgamma3(p = temp, scale = 1, shape = 5, threshold = 0)
```

---

plnorm3 *plnorm3: The Lognormal Distribution (3 Parameter)*

Description

Density function, distribution function, and quantile function for the Lognormal distribution.

Usage

```
plnorm3(q, meanlog, sdlog, threshold)
```

Arguments

q	A numeric vector of quantiles.
meanlog, sdlog	The mean and standard deviation of the distribution on the log scale with default values of 0 and 1 respectively.
threshold	The threshold parameter, default is 0.

Details

The Lognormal distribution with meanlog parameter zeta, sdlog parameter sigma, and threshold parameter theta has a density given by:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma(x-\theta)} \exp\left(-\frac{(\log(x-\theta)-\zeta)^2}{2\sigma^2}\right)$$

The cumulative distribution function is given by:

$$F(x) = \Phi\left(\frac{\log(x-\theta)-\zeta}{\sigma}\right)$$

where Φ is the cumulative distribution function of the standard normal distribution.

Value

dlnorm3 gives the density, plnorm3 gives the distribution function, and qlnorm3 gives the quantile function.

Examples

```
dlnorm3(x = 2, meanlog = 0, sdlog = 1/8, threshold = 1)
temp <- plnorm3(q = 2, meanlog = 0, sdlog = 1/8, threshold = 1)
temp
qlnorm3(p = temp, meanlog = 0, sdlog = 1/8, threshold = 1)
```

---

ppPlot — *ppPlot: Probability Plots for various distributions*

Description

Function ppPlot creates a Probability plot of the values in x including a line.

Usage

```
ppPlot(
  x,
  distribution,
  confbounds = TRUE,
  alpha,
  probs,
  main,
  xlab,
  ylab,
  xlim,
  ylim,
  border = "red",
  bounds.col = "black",
  bounds.lty = 1,
  start,
  showPlot = TRUE,
  axis.y.right = FALSE,
  bw.theme = FALSE
)
```

Arguments

x	Numeric vector containing the sample data for the ppPlot.
distribution	Character string specifying the distribution of x. The function ppPlot will support the following character strings for distribution: • `beta`

- `cauchy`
- `chi-squared`
- `exponential`
- `f`
- `gamma`
- `geometric`
- `log-normal`
- `lognormal`
- `logistic`
- `negative binomial`
- `normal`
- `Poisson`
- `weibull`

By default `distribution` is set to `normal`.

`confbounds` Logical value: whether to display confidence bounds. Default is `TRUE`.

`alpha` Numeric value: significance level for confidence bounds, default is '0.05'.

`probs` Vector containing the percentages for the y axis. All the values need to be between '0' and '1'. If 'probs' is missing it will be calculated internally.

`main` Character string: title of the plot.

`xlab` Character string: label for the x-axis.

`ylab` Character string: label for the y-axis.

`xlim` Numeric vector of length 2: limits for the x-axis.

`ylim` Numeric vector of length 2: limits for the y-axis.

`border` Character or numeric: color for the border of the line through the quantiles. Default is `red`.

`bounds.col` Character or numeric: color for the confidence bounds lines. Default is `black`.

`bounds.lty` Numeric or character: line type for the confidence bounds lines. This can be specified with either an integer (0-6) or a name:

- 0: blank
- 1: solid
- 2: dashed
- 3: dotted
- 4: dotdash
- 5: longdash
- 6: twodash

Default is '1' (solid line).

`start` A named list giving the parameters to be fitted with initial values. Must be supplied for some distributions (see Details).

`showPlot` Logical value indicating whether to display the plot. By default, `showPlot` is set to `TRUE`.

`axis.y.right` Logical value indicating whether to display the y-axis on the right side. By default, `axis.y.right` is set to `FALSE`.

`bw.theme` Logical value indicating whether to use a black-and-white theme from the `ggplot2` package for the plot. By default, `bw.theme` is set to `FALSE`.

Details

Distribution fitting is performed using the `FitDistr` function from this package. For the computation of the confidence bounds, the variance of the quantiles is estimated using the delta method, which involves the estimation of the observed Fisher Information matrix as well as the gradient of the CDF of the fitted distribution. Where possible, those values are replaced by their normal approximation.

Value

The function `ppPlot` returns an invisible list containing:

`x`	x coordinates.
`y`	y coordinates.
`int`	Intercept.
`slope`	Slope.
`plot`	The generated PP plot.

See Also

`qqPlot`, `FitDistr`.

Examples

```
set.seed(123)
ppPlot(rnorm(20, mean=90, sd=5), "normal",alpha=0.30)
ppPlot(rcauchy(100), "cauchy")
ppPlot(rweibull(50, shape = 1, scale = 1), "weibull")
ppPlot(rlogis(50), "logistic")
ppPlot(rlnorm(50) , "log-normal")
ppPlot(rbeta(10, 0.7, 1.5),"beta")
ppPlot(rpois(20,3), "poisson")
ppPlot(rchisq(20, 10),"chi-squared")
ppPlot(rgeom(20, prob = 1/4), "geometric")
ppPlot(rnbinom(n = 20, size = 3, prob = 0.2), "negative binomial")
ppPlot(rf(20, df1 = 10, df2 = 20), "f")
```

`print_adtest` *print_adtest: Test Statistics*

Description

Function to show `adtest`.

Usage

```
print_adtest(x, digits = 4, quote = TRUE, prefix = "", ...)
```

Arguments

x	Needs to be an object of class adtest.
digits	Minimal number of significant digits.
quote	Logical, indicating whether or not strings should be printed with surrounding quotes. By default quote is set to TRUE.
prefix	Single character or character string that will be printed in front of x.
...	Further arguments passed to or from other methods.

Value

The function returns a summary of Anderson Darling Test

Examples

```
data <- rnorm(20, mean = 20)
pcr1<-pcr(data, "normal", lsl = 17, usl = 23, plot = FALSE)
print_adtest(pcr1$adTest)
```

---

pweibull3 — *pweibull3: The Weibull Distribution (3 Parameter)*

Description

Density function, distribution function, and quantile function for the Weibull distribution with a threshold parameter.

Usage

```
pweibull3(q, shape, scale, threshold)
```

Arguments

q	A numeric vector of quantiles.
shape	The shape parameter of the Weibull distribution. Default is 1.
scale	The scale parameter of the Weibull distribution. Default is 1.
threshold	The threshold (or location) parameter of the Weibull distribution. Default is 0.

Details

The Weibull distribution with the scale parameter alpha, shape parameter c, and threshold parameter zeta has a density function given by:

$$f(x) = \frac{c}{\alpha}\left(\frac{x-\zeta}{\alpha}\right)^{c-1}\exp\left(-\left(\frac{x-\zeta}{\alpha}\right)^{c}\right)$$

The cumulative distribution function is given by:

$$F(x) = 1 - \exp\left(-\left(\frac{x-\zeta}{\alpha}\right)^{c}\right)$$

Value

dweibull3 returns the density, pweibull3 returns the distribution function, and qweibull3 returns the quantile function for the Weibull distribution with a threshold.

Examples

```
dweibull3(x = 1, scale = 1, shape = 5, threshold = 0)
temp <- pweibull3(q = 1, scale = 1, shape = 5, threshold = 0)
temp
qweibull3(p = temp, scale = 1, shape = 5, threshold = 0)
```

---

qgamma3 *qgamma3: The gamma Distribution (3 Parameter)*

Description

Density function, distribution function, and quantile function for the Gamma distribution.

Usage

```
qgamma3(p, shape, scale, threshold, ...)
```

Arguments

p	A numeric vector of probabilities.
shape	The shape parameter, default is 1.
scale	The scale parameter, default is 1.
threshold	The threshold parameter, default is 0.
...	Additional arguments that can be passed to uniroot.

Details

The Gamma distribution with scale parameter alpha, shape parameter c, and threshold parameter zeta has a density given by:

$$f(x) = \frac{c}{\alpha}\left(\frac{x-\zeta}{\alpha}\right)^{c-1}\exp\left(-\left(\frac{x-\zeta}{\alpha}\right)^{c}\right)$$

The cumulative distribution function is given by:

$$F(x) = 1 - \exp\left(-\left(\frac{x-\zeta}{\alpha}\right)^{c}\right)$$

Value

dgamma3 gives the density, pgamma3 gives the distribution function, and qgamma3 gives the quantile function.

Examples

```
dgamma3(x = 1, scale = 1, shape = 5, threshold = 0)
temp <- pgamma3(q = 1, scale = 1, shape = 5, threshold = 0)
temp
qgamma3(p = temp, scale = 1, shape = 5, threshold = 0)
```

---

qlnorm3 *qlnorm3: The Lognormal Distribution (3 Parameter)*

Description

Density function, distribution function, and quantile function for the Lognormal distribution.

Usage

```
qlnorm3(p, meanlog, sdlog, threshold, ...)
```

Arguments

`p`	A numeric vector of probabilities.
`meanlog`, `sdlog`	The mean and standard deviation of the distribution on the log scale with default values of `0` and `1` respectively.
`threshold`	The threshold parameter, default is `0`.
`...`	Additional arguments that can be passed to `uniroot`.

Details

The Lognormal distribution with `meanlog` parameter zeta, `sdlog` parameter sigma, and `threshold` parameter theta has a density given by:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma(x-\theta)} \exp\left(-\frac{(\log(x-\theta)-\zeta)^2}{2\sigma^2}\right)$$

The cumulative distribution function is given by:

$$F(x) = \Phi\left(\frac{\log(x-\theta)-\zeta}{\sigma}\right)$$

where Φ is the cumulative distribution function of the standard normal distribution.

Value

`dlnorm3` gives the density, `plnorm3` gives the distribution function, and `qlnorm3` gives the quantile function.

Examples

```
dlnorm3(x = 2, meanlog = 0, sdlog = 1/8, threshold = 1)
temp <- plnorm3(q = 2, meanlog = 0, sdlog = 1/8, threshold = 1)
temp
qlnorm3(p = temp, meanlog = 0, sdlog = 1/8, threshold = 1)
```

---

qqPlot *qqPlot: Quantile-Quantile Plots for various distributions*

Description

Function qqPlot creates a QQ plot of the values in x including a line which passes through the first and third quartiles.

Usage

```
qqPlot(
  x,
  y,
  confbounds = TRUE,
  alpha,
  main,
  xlab,
  ylab,
  xlim,
  ylim,
  border = "red",
  bounds.col = "black",
  bounds.lty = 1,
  start,
  showPlot = TRUE,
  axis.y.right = FALSE,
  bw.theme = FALSE
)
```

Arguments

x — The sample for qqPlot.

y — Character string specifying the distribution of x. The function qqPlot supports the following character strings for y:

- `beta`
- `cauchy`
- `chi-squared`
- `exponential`
- `f`

- `gamma`
- `geometric`
- `log-normal`
- `lognormal`
- `logistic`
- `negative binomial`
- `normal`
- `Poisson`
- `weibull`

By default `distribution` is set to `normal`.

`confbounds` Logical value indicating whether to display confidence bounds. By default, `confbounds` is set to `TRUE`.

`alpha` Numeric value specifying the significance level for the confidence bounds, set to '0.05' by default.

`main` A character string for the main title of the plot.

`xlab` A character string for the x-axis label.

`ylab` A character string for the y-axis label.

`xlim` A numeric vector of length 2 to specify the limits of the x-axis.

`ylim` A numeric vector of length 2 to specify the limits of the y-axis.

`border` A numerical value or single character string giving the color of the interpolation line. By default, `border` is set to `red`.

`bounds.col` A numerical value or single character string giving the color of the confidence bounds lines. By default, `bounds.col` is set to `black`.

`bounds.lty` A numeric or character: line type for the confidence bounds lines. This can be specified with either an integer (0-6) or a name:

- 0: blank
- 1: solid
- 2: dashed
- 3: dotted
- 4: dotdash
- 5: longdash
- 6: twodash

Default is '1' (solid line).

`start` A named list giving the parameters to be fitted with initial values. Must be supplied for some distributions (see Details).

`showPlot` Logical value indicating whether to display the plot. By default, `showPlot` is set to `TRUE`.

`axis.y.right` Logical value indicating whether to display the y-axis on the right side. By default, `axis.y.right` is set to `FALSE`.

`bw.theme` Logical value indicating whether to use a black-and-white theme from the `ggplot2` package for the plot. By default, `bw.theme` is set to `FALSE`.

Details

Distribution fitting is performed using the `FitDistr` function from this package. For the computation of the confidence bounds, the variance of the quantiles is estimated using the delta method, which involves the estimation of the observed Fisher Information matrix as well as the gradient of the CDF of the fitted distribution. Where possible, those values are replaced by their normal approximation.

Value

The function `qqPlot` returns an invisible list containing:

`x`	Sample quantiles.
`y`	Theoretical quantiles.
`int`	Intercept of the fitted line.
`slope`	Slope of the fitted line.
`plot`	The generated QQ plot.

See Also

`ppPlot`, `FitDistr`.

Examples

```
set.seed(123)
qqPlot(rnorm(20, mean=90, sd=5), "normal",alpha=0.30)
qqPlot(rcauchy(100), "cauchy")
qqPlot(rweibull(50, shape = 1, scale = 1), "weibull")
qqPlot(rlogis(50), "logistic")
qqPlot(rlnorm(50) , "log-normal")
qqPlot(rbeta(10, 0.7, 1.5),"beta")
qqPlot(rpois(20,3), "poisson")
qqPlot(rchisq(20, 10),"chi-squared")
qqPlot(rgeom(20, prob = 1/4), "geometric")
qqPlot(rnbinom(n = 20, size = 3, prob = 0.2), "negative binomial")
qqPlot(rf(20, df1 = 10, df2 = 20), "f")
```

`qweibull3` *qweibull3: The Weibull Distribution (3 Parameter)*

Description

Density function, distribution function, and quantile function for the Weibull distribution with a threshold parameter.

Usage

```
qweibull3(p, shape, scale, threshold, ...)
```

Arguments

`p`	A numeric vector of probabilities.
`shape`	The shape parameter of the Weibull distribution. Default is 1.
`scale`	The scale parameter of the Weibull distribution. Default is 1.
`threshold`	The threshold (or location) parameter of the Weibull distribution. Default is 0.
`...`	Additional arguments passed to `uniroot` for `qweibull3`.

Details

The Weibull distribution with the `scale` parameter alpha, `shape` parameter c, and `threshold` parameter zeta has a density function given by:

$$f(x) = \frac{c}{\alpha}\left(\frac{x-\zeta}{\alpha}\right)^{c-1}\exp\left(-\left(\frac{x-\zeta}{\alpha}\right)^{c}\right)$$

The cumulative distribution function is given by:

$$F(x) = 1 - \exp\left(-\left(\frac{x-\zeta}{\alpha}\right)^{c}\right)$$

Value

`dweibull3` returns the density, `pweibull3` returns the distribution function, and `qweibull3` returns the quantile function for the Weibull distribution with a threshold.

Examples

```
dweibull3(x = 1, scale = 1, shape = 5, threshold = 0)
temp <- pweibull3(q = 1, scale = 1, shape = 5, threshold = 0)
temp
qweibull3(p = temp, scale = 1, shape = 5, threshold = 0)
```

`randomize` *randomize: Randomization*

Description

Function to do randomize the run order of factorial designs.

Usage

```
randomize(fdo, random.seed = 93275938, so = FALSE)
```

Arguments

fdo	An object of class facDesign.c.
random.seed	Seed for randomness.
so	Logical value specifying whether the standard order should be used or not. By default so is set to FALSE.

Value

An object of class facDesign.c with the run order randomized.

Examples

```
dfrac <- fracDesign(k = 3)
randomize(dfrac)
```

---

rsmChoose *rsmChoose: Choosing a response surface design from a table*

Description

Designs displayed are central composite designs with orthogonal blocking and near rotatability. The function allows users to choose a design by clicking with the mouse into the appropriate field.

Usage

```
rsmChoose()
```

Value

Returns an object of class facDesign.c.

See Also

fracChoose, rsmDesign

Examples

```
rsmChoose()
```

rsmDesign *rsmDesign: Generate a response surface design.*

Description

Generates a response surface design containing a cube, centerCube, star, and centerStar portion.

Usage

```
rsmDesign(
  k = 3,
  p = 0,
  alpha = "rotatable",
  blocks = 1,
  cc = 1,
  cs = 1,
  fp = 1,
  sp = 1,
  faceCentered = FALSE
)
```

Arguments

k	Integer value giving the number of factors. By default, k is set to '3'.
p	Integer value giving the number of additional factors in the response surface design by aliasing effects. Default is '0'.
alpha	Character string indicating the type of star points to generate. Should be `rotatable` (default), `orthogonal`, or `both`. If `both`, values for cc and cs will be discarded.
blocks	Integer value specifying the number of blocks in the response surface design. Default is '1'.
cc	Integer value giving the number of centerpoints (per block) in the cube portion (i.e., the factorial 2^k design) of the response surface design. Default is '1'.
cs	Integer value specifying the number of centerpoints in the star portion. Default is '1'.
fp	Integer value giving the number of replications per factorial point (i.e., corner points). Default is '1'.
sp	Integer value specifying the number of replications per star point. Default is '1'.
faceCentered	Logical value indicating whether to use a faceCentered response surface design (i.e., alpha = '1'). Default is FALSE.

Details

Generated designs consist of a cube, centerCube, star, and centerStar portion. The replication structure can be set with the parameters cc (centerCube), cs (centerStar), fp (factorialPoints), and sp (starPoints).

Value

The function returns an object of class facDesign.c.

See Also

facDesign, fracDesign, fracChoose, pbDesign, rsmChoose

Examples

```
# Example 1: Central composite design for 2 factors with 2 blocks, alpha = 1.41,
# 5 centerpoints in the cube portion and 3 centerpoints in the star portion:
rsmDesign(k = 2, blocks = 2, alpha = sqrt(2), cc = 5, cs = 3)

# Example 2: Central composite design with both, orthogonality and near rotatability
rsmDesign(k = 2, blocks = 2, alpha = "both")

# Example 3: Central composite design with:
# 2 centerpoints in the factorial portion of the design (i.e., 2)
# 1 centerpoint in the star portion of the design (i.e., 1)
# 2 replications per factorial point (i.e., 2^3*2 = 16)
# 3 replications per star point (i.e., 3*2*3 = 18)
# Makes a total of 37 factor combinations
rsdo = rsmDesign(k = 3, blocks = 1, alpha = 2, cc = 2, cs = 1, fp = 2, sp = 3)
```

---

simProc — *simProc: Simulated Process*

Description

This is a function to simulate a black box process for teaching the use of designed experiments. The optimal factor settings can be found using a sequential assembly strategy i.e. apply a 2^k factorial design first, calculate the path of the steepest ascent, again apply a 2^k factorial design and augment a star portion to find the optimal factor settings. Of course, other strategies are possible.

Usage

```
simProc(x1, x2, x3, noise = TRUE)
```

Arguments

x1	numeric vector containing the values for factor 1.
x2	numeric vector containing the values for factor 2.
x3	numeric vector containing the values for factor 3.
noise	logical value deciding whether noise should be added or not. Default setting is TRUE.

Value

simProc returns a numeric value within the range [0,1].

Examples

```
simProc(120, 140, 1)
simProc(120, 220, 1)
simProc(160, 140, 1)
```

---

snPlot *snPlot: Signal-to-Noise-Ratio Plots*

Description

Creates a Signal-to-Noise Ratio plot for designs of type taguchiDesign.c with at least two replicates.

Usage

```
snPlot(object, type = "nominal", factors, fun = mean, response = NULL,
       points = FALSE, classic = FALSE, lty, xlab, ylab,
       main, ylim, l.col, p.col, ld.col, pch)
```

Arguments

object	An object of class taguchiDesign.c.
type	A character string specifying the type of the Signal-to-Noise Ratio plot. Possible values are: • `nominal`: Nominal-the-best plot to equalize observed values to a nominal value. • `smaller`: Smaller-the-better plot to minimize observed values. • `larger`: Larger-the-better plot to maximize observed values. Default is `nominal`.
factors	The factors for which the effect plot is to be created.
fun	A function for constructing the effect plot such as mean, median, etc. Default is mean.
response	A character string specifying the response variable. If object contains multiple responses, this parameter selects one column to plot. Default is NULL.
points	A logical value. If TRUE, points are shown in addition to values derived from fun. Default is FALSE.
classic	A logical value. If TRUE, creates an effect plot as depicted in most textbooks. Default is FALSE.
lty	A numeric value specifying the line type to be used.
xlab	A title for the x-axis.

`ylab`	A title for the y-axis.
`main`	An overall title for the plot.
`ylim`	A numeric vector of length 2 specifying the limits of the y-axis.
`l.col`	A color for the lines.
`p.col`	A color for the points.
`ld.col`	A color for the dashed line.
`pch`	The symbol for plotting points.

Details

The Signal-to-Noise Ratio (SNR) is calculated based on the type specified:

- `` `nominal` ``:
$$SN = 10 \cdot log(mean(y)/var(y))$$
- `` `smaller` ``:
$$SN = -10 \cdot log((1/n) \cdot sum(y^2))$$
- `` `larger` ``:
$$SN = -10 \cdot log((1/n) \cdot sum(1/y^2))$$

Signal-to-Noise Ratio plots are used to estimate the effects of individual factors and to judge the variance and validity of results from an effect plot.

Value

An invisible `data.frame` containing all the single Signal-to-Noise Ratios.

Examples

```
tdo <- taguchiDesign("L9_3", replicates = 3)
tdo$.response(rnorm(27))
snPlot(tdo, points = TRUE, l.col = 2, p.col = 2, ld.col = 2, pch = 16, lty = 3)
```

`starDesign` *starDesign: Axial Design*

Description

`starDesign` is a function to create the star portion of a response surface design. The starDesign function can be used to create a star portion of a response surface design for a sequential assembly strategy. One can either specify k and p and alpha and cs and cc OR simply simply pass an object of class `facDesign.c` to the data. In the latter an object of class `facDesign.c` otherwise a list containing the axial runs and centerpoints is returned.

Usage

```
starDesign(
  k,
  p = 0,
  alpha = c("both", "rotatable", "orthogonal"),
  cs,
  cc,
  data
)
```

Arguments

k	Integer value giving number of factors.
p	Integer value giving the number of factors via aliasing. By default set to '0'.
alpha	If no numeric value is given defaults to `both` i.e. `orthogonality` and `rotatibility` which can be set as character strings too.
cs	Integer value giving the number of centerpoints in the star portion of the design.
cc	Integer value giving the number of centerpoints in the cube portion of the design.
data	Optional. An object of class facDesign.c.

Value

starDesign returns a facDesign.c object if an object of class facDesign.c is given or a list containing entries for axial runs and center points in the cube and the star portion of a design.

See Also

facDesign, fracDesign, rsmDesign, mixDesign

Examples

```
# Example 1: sequential assembly
# Factorial design with one center point in the cube portion
fdo = facDesign(k = 3, centerCube = 1)
# Set the response via generic response method
fdo$.response(1:9)
# Sequential assembly of a response surface design (rsd)
rsd = starDesign(data = fdo)

# Example 2: Returning a list of star point designs
starDesign(k = 3, cc = 2, cs = 2, alpha = "orthogonal")
starDesign(k = 3, cc = 2, cs = 2, alpha = "rotatable")
starDesign(k = 3, cc = 2, cs = 2, alpha = "both")
```

steepAscent *steepAscent: Steepest Ascent*

Description

steepAscent is a method to calculate the steepest ascent for a facDesign.c object.

Usage

```
steepAscent(factors, response, size = 0.2, steps = 5, data)
```

Arguments

factors	List containing vector of factor names (coded) to be included in calculation, first factor is the reference factor.
response	A character of response given in data.
size	Numeric integer value giving the step size in coded units for the first factor given in factors. By default size is set to 0.2.
steps	Numeric integer value giving the number of steps. By default step is set to '5'.
data	An object of class facDesign.c.

Value

steepAscent returns an object of class steepAscent.c.

See Also

optimum, desirability

Examples

```
# Example 1
fdo = facDesign(k = 2, centerCube = 5)
fdo$lows(c(170, 150))
fdo$highs(c(230, 250))
fdo$names(c("temperature", "time"))
fdo$unit(c("C", "minutes"))
yield = c(32.79, 24.07, 48.94, 52.49, 38.89, 48.29, 29.68, 46.5, 44.15)
fdo$.response(yield)
fdo$summary()

sao = steepAscent(factors = c("B", "A"), response = "yield", size = 1,
                  data = fdo)
```

steepAscent.c *steepAscent-class: Class ‘steepAscent‘*

Description

The `steepAscent.c` class represents a steepest ascent algorithm in a factorial design context. This class is used for optimizing designs based on iterative improvements.

Public fields

`name` A character string representing the name of the steep ascent design.

`X` A data frame containing the design matrix for the steepest ascent procedure. This matrix represents the factors and their levels at each iteration.

`response` A data frame containing the response values associated with the design matrix.

Methods

Public methods:

- `steepAscent.c$.response()`
- `steepAscent.c$get()`
- `steepAscent.c$as.data.frame()`
- `steepAscent.c$print()`
- `steepAscent.c$plot()`
- `steepAscent.c$clone()`

Method `.response()`**:** Get and set the ‘response‘ values in an object of class ‘steepAscent.c‘.

Usage:

```
steepAscent.c$.response(value)
```

Arguments:

`value` A data frame or numeric vector to set as the new ‘response‘. If missing, returns the current ‘response‘.

Method `get()`**:** Access specific elements in the design matrix or response data of the object.

Usage:

```
steepAscent.c$get(i, j)
```

Arguments:

`i` An integer specifying the row index to retrieve.

`j` An integer specifying the column index to retrieve.

Method `as.data.frame()`**:** Convert the object to a data frame.

Usage:

```
steepAscent.c$as.data.frame()
```

Method `print()`**:** Print the details of the object.

Usage:

```
steepAscent.c$print()
```

Method `plot()`**:** Plot the results of the steepest ascent procedure for an object of class 'steepAscent.c'.

Usage:

```
steepAscent.c$plot(main, xlab, ylab, l.col, p.col, line.type, point.shape)
```

Arguments:

`main` The main title of the plot.

`xlab` The label for the x-axis.

`ylab` The label for the y-axis.

`l.col` Color for the line in the plot.

`p.col` Color for the points in the plot.

`line.type` Type of the line used in the plot.

`point.shape` Shape of the points used in the plot.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
steepAscent.c$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`steepAscent`, `desirability.c`, `optimum`

`summaryFits` *summaryFits: Fit Summary*

Description

Function to provide an overview of fitted linear models for objects of class `facDesign.c`.

Usage

```
summaryFits(fdo, lmFit = TRUE, curvTest = TRUE)
```

Arguments

`fdo`	An object of class `facDesign.c`.
`lmFit`	A logical value deciding whether the fits from the object `fdo` should be included or not. By default, `lmFit` is set to `TRUE`.
`curvTest`	A logical value deciding whether curvature tests should be performed or not. By default, `curvTest` is set to `TRUE`.

Value

A summary output of the fitted linear models, which may include the linear fits, curvature tests, and original fit values, depending on the input parameters.

Examples

```
dfac <- facDesign(k = 3)
dfac$.response(data.frame(y = rnorm(8), y2 = rnorm(8)))
dfac$set.fits(lm(y ~ A + B , data = dfac$as.data.frame()))
dfac$set.fits(lm(y2 ~ A + C, data = dfac$as.data.frame()))
summaryFits(dfac)
```

---

taguchiChoose *taguchiChoose: Taguchi Designs*

Description

Shows a matrix of possible taguchi designs

Usage

```
taguchiChoose(
  factors1 = 0,
  factors2 = 0,
  level1 = 0,
  level2 = 0,
  ia = 0,
  col = 2,
  randomize = TRUE,
  replicates = 1
)
```

Arguments

factors1	Integer number of factors on level1. By default set to ‘0‘.
factors2	Integer number of factors on level2. By default set to ‘0‘.
level1	Integer number of levels on level1. By default set to ‘0‘.
level2	Integer number of levels on level2. By default set to ‘0‘.
ia	Integer number of interactions. By default set to ‘0‘.
col	Select the color scheme for the selection matrix: use 1 for blue, 2 for pink (default), and 3 for a variety of colors.
randomize	A logical value (TRUE/FALSE) that specifies whether to randomize the RunOrder of the design. By default, randomize is set to TRUE.
replicates	An integer specifying the number of replicates for each run in the design.

Details

taguchiChoose returns possible taguchi designs. Specifying the number of factor1 factors with level1 levels (factors1 = 2, level1 = 3 means 2 factors with 3 factor levels) and factor2 factors with level2 levels and desired interactions one or more taguchi designs are suggested. If all parameters are set to 0, a matrix of possible taguchi designs is shown.

Value

taguchiChoose returns an object of class taguchiDesign.

See Also

- facDesign: for 2^k factorial designs.
- rsmDesign: for response surface designs.
- fracDesign: for fractional factorial design.
- gageRRDesign: for gage designs.

Examples

```
tdo1 <- taguchiChoose()
tdo1 <- taguchiChoose(factors1 = 3, level1 = 2)
```

---

taguchiDesign *taguchiDesign: Taguchi Designs*

Description

Function to create a taguchi design.

Usage

```
taguchiDesign(design, randomize = TRUE, replicates = 1)
```

Arguments

design — A character string specifying the orthogonal array of the Taguchi design. The available options are:

- ‘L4_2" for three two-level factors.
- ‘L8_2" for seven two-level factors.
- ‘L9_3" for four three-level factors.
- ‘L12_2" for 11 two-level factors.
- ‘L16_2" for 16 two-level factors
- ‘L16_4" for 16 four-level factors.
- ‘L18_2_3" for one two-level and seven three-level factors.
- ‘L25_5" for six five-level factors.

- ‘L27_3" for 13 three-level factors.
- ‘L32_2" for 32 two-level factors.
- ‘L32_2_4" for one two-level factor and nine four-level factors.
- ‘L36_2_3_a" for 11 two-level factors and 12 three-level factors.
- ‘L36_2_3_b" for three two-level factors and 13 three-level factors.
- ‘L50_2_5" for one two-level factor and eleven five-level factors.
- ‘L8_4_2" for one four-level factor and four two-level factors.
- ‘L16_4_2_a" for one four-level factor and 12 two-level factors.
- ‘L16_4_2_b" for two four-level factors and nine two-level factors.
- ‘L16_4_2_c" for three four-level factors and six two-level factors.
- ‘L16_4_2_d" for five four-level factors and two two-level factors.
- ‘L18_6_3" for one six-level factor and six three-level factors.

`randomize` A logical value (`TRUE`/`FALSE`) that specifies whether to randomize the RunOrder of the design. By default, `randomize` is set to `TRUE`.

`replicates` An integer specifying the number of replicates for each run in the design.

Details

An overview of possible taguchi designs is possible with `taguchiChoose`.

Value

A `taguchiDesign` returns an object of class `taguchiDesign`.

See Also

- `facDesign`: for 2^k factorial designs.
- `rsmDesign`: for response surface designs.
- `fracDesign`: for fractional factorial design.
- `pbDesign`: for response surface designs.
- `gageRRDesign`: for gage designs.

Examples

```
tdo <- taguchiDesign("L9_3")
tdo$values(list(A = c("material 1", "material 2", "material 3"), B = c(29, 30, 35)))
tdo$names(c("Factor 1", "Factor 2", "Factor 3", "Factor 4"))
tdo$.response(rnorm(9))
tdo$summary()
```

taguchiDesign.c *taguchiDesign*

Description

An R6 class representing a Taguchi experimental design.

Public fields

`name` A character string specifying the name of the design. Default is `NULL`.

`factors` A list of factors included in the Taguchi design. Each factor is typically an instance of the `taguchiFactor` class.

`design` A ‘data.frame‘ representing the design matrix of the experiment. This includes the levels of each factor for every run of the experiment. Default is an empty `data.frame`.

`designType` A character string specifying the type of Taguchi design used. Default is `NULL`.

`replic` A ‘data.frame‘ containing the replication information for the design. Default is an empty `data.frame`.

`response` A ‘data.frame‘ storing the response values collected from the experiment. Default is an empty `data.frame`.

`Type` A ‘data.frame‘ specifying the type of responses or factors involved in the design. Default is an empty `data.frame`.

`block` A ‘data.frame‘ indicating any blocking factors used in the design. Default is an empty `data.frame`.

`runOrder` A ‘data.frame‘ detailing the order in which the experimental runs were conducted. Default is an empty `data.frame`.

`standardOrder` A ‘data.frame‘ detailing the standard order of the experimental runs. Default is an empty `data.frame`.

`desireVal` A list storing desired values for responses in the experiment. Default is an empty list.

`desirability` A list storing desirability functions used to evaluate the outcomes of the experiment. Default is an empty list.

`fits` A ‘data.frame‘ containing model fits or other statistical summaries from the analysis of the experimental data. Default is an empty `data.frame`.

Methods

Public methods:

- `taguchiDesign.c$values()`
- `taguchiDesign.c$units()`
- `taguchiDesign.c$.factors()`
- `taguchiDesign.c$names()`
- `taguchiDesign.c$as.data.frame()`
- `taguchiDesign.c$print()`

- taguchiDesign.c$.response()
- taguchiDesign.c$.nfp()
- taguchiDesign.c$summary()
- taguchiDesign.c$effectPlot()
- taguchiDesign.c$identity()
- taguchiDesign.c$clone()

Method `values()`**:** Get and set the `values` for an object of class `taguchiDesign`.

Usage:

```
taguchiDesign.c$values(value)
```

Arguments:

`value` New value, If missing value get the `values`.

Method `units()`**:** Get and set the `units` for an object of class `taguchiDesign`.

Usage:

```
taguchiDesign.c$units(value)
```

Arguments:

`value` New units, If missing value get the `units`.

Method `.factors()`**:** Get and set the `factors` in an object of class `taguchiDesign`.

Usage:

```
taguchiDesign.c$.factors(value)
```

Arguments:

`value` New factors, If missing value get the `factors`.

Method `names()`**:** Get and set the `names` in an object of class `taguchiDesign`.

Usage:

```
taguchiDesign.c$names(value)
```

Arguments:

`value` New names, If missing value get the `names`.

Method `as.data.frame()`**:** Return a data frame with the information of the object `taguchiDesign.c`.

Usage:

```
taguchiDesign.c$as.data.frame()
```

Method `print()`**:** Methods for function `print` in Package `base`.

Usage:

```
taguchiDesign.c$print()
```

Method `.response()`**:** Get and set the the `response` in an object of class `taguchiDesign`.

Usage:

```
taguchiDesign.c$.response(value)
```

Arguments:

`value` New response, If missing value get the `response`.

Method `.nfp()`**:** Prints a summary of the factors attributes including their low, high, name, unit, and type.

Usage:

```
taguchiDesign.c$.nfp()
```

Method `summary()`**:** Methods for function `summary` in Package `base`.

Usage:

```
taguchiDesign.c$summary()
```

Method `effectPlot()`**:** Plots the effects of factors on the response variables.

Usage:

```
taguchiDesign.c$effectPlot(
  factors,
  fun = mean,
  response = NULL,
  points = FALSE,
  l.col,
  p.col,
  ld.col,
  lty,
  xlab,
  ylab,
  main,
  ylim,
  pch
)
```

Arguments:

`factors` Factors to be plotted.

`fun` Function applied to the response variables (e.g., mean).

`response` Optional; specifies which response variables to plot.

`points` Logical; if TRUE, plots data points.

`l.col` A color for the lines.

`p.col` A color for the points.

`ld.col` A color for the dashed line.

`lty` Line type for plotting.

`xlab` Label for the x-axis.

`ylab` Label for the y-axis.

`main` Main title for the plot.

`ylim` Limits for the y-axis.

`pch` The symbol for plotting points.

Examples:

```
tdo = taguchiDesign("L9_3")
tdo$.response(rnorm(9))
tdo$effectPlot(points = TRUE, pch = 16, lty = 3)
```

Method `identity()`**:** Calculates the alias table for a fractional factorial design and prints an easy to read summary of the defining relations such as 'I = ABCD' for a standard 2^(4-1) factorial design.

Usage:

```
taguchiDesign.c$identity()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
taguchiDesign.c$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `taguchiDesign.c$effectPlot`
## ------------------------------------------------

tdo = taguchiDesign("L9_3")
tdo$.response(rnorm(9))
tdo$effectPlot(points = TRUE, pch = 16, lty = 3)
```

`taguchiFactor` *taguchiFactor*

Description

An R6 class representing a factor in a Taguchi design.

Public fields

`values` A vector containing the levels or values associated with the factor. Default is `NA`.

`name` A character string specifying the name of the factor. Default is an empty string ``.

`unit` A character string specifying the unit of measurement for the factor. Default is an empty string ``.

`type` A character string specifying the type of the factor, which can be either `numeric` or `categorical`. Default is `numeric`.

Methods

Public methods:

- `taguchiFactor$attributes()`
- `taguchiFactor$.values()`
- `taguchiFactor$.unit()`
- `taguchiFactor$names()`

- taguchiFactor$clone()

Method `attributes()`**:** Get the attributes of the factor.

Usage:

```
taguchiFactor$attributes()
```

Method `.values()`**:** Get and set the `values` for the factors in an object of class `taguchiFactor`.

Usage:

```
taguchiFactor$.values(value)
```

Arguments:

`value` New values, If missing value get the `values`.

Method `.unit()`**:** Get and set the `units` for the factors in an object of class `taguchiFactor`.

Usage:

```
taguchiFactor$.unit(value)
```

Arguments:

`value` New unit, If missing value get the `units`.

Method `names()`**:** Get and set the `names` in an object of class `taguchiFactor`.

Usage:

```
taguchiFactor$names(value)
```

Arguments:

`value` New names, If missing value get the `names`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
taguchiFactor$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`wirePlot` *wirePlot: 3D Plot*

Description

Creates a wireframe diagram for an object of class `facDesign.c`.

Usage

```
wirePlot(
  x,
  y,
  z,
  data = NULL,
  xlim,
  ylim,
  zlim,
  main,
  xlab,
  ylab,
  sub,
  sub.a = TRUE,
  zlab,
  form = "fit",
  col = "Rainbow",
  steps,
  fun,
  plot = TRUE,
  show.scale = TRUE,
  n.scene = "scene"
)
```

Arguments

`x`	Name providing the Factor A for the plot.
`y`	Name providing the Factor B for the plot.
`z`	Name giving the Response variable.
`data`	Needs to be an object of class `facDesign` and contains the names of `x`, `y`, `z`.
`xlim`	Numeric vector of length 2: limits for the x-axis. If missing, limits are set automatically.
`ylim`	Numeric vector of length 2: limits for the y-axis. If missing, limits are set automatically.
`zlim`	Numeric vector of length 2: limits for the z-axis. If missing, limits are set automatically.
`main`	Character string: title of the plot.
`xlab`	Character string: label for the x-axis.
`ylab`	Character string: label for the y-axis.
`sub`	Character string: subtitle for the plot. Default is `NULL`.
`sub.a`	Logical value indicating whether to display the subtitle. Default is `TRUE`.
`zlab`	Character string: label for the z-axis.
`form`	Character string specifying the form of the surface to be plotted. Options include • `` `quadratic` ``

- `full`
- `interaction`
- `linear`
- `fit`

Default is `fit`.

col	Character string specifying the color palette to use for the plot (e.g., `Rainbow`, `Jet`, `Earth`, `Electric`). Default is `Rainbow`.
steps	Numeric value specifying the number of steps for the grid in the plot. Higher values result in a smoother surface.
fun	Optional function to be applied to the data before plotting.
plot	Logical value indicating whether to display the plot. Default is TRUE.
show.scale	Logical value indicating whether to display the color scale on the plot. Default is TRUE.
n.scene	Character string specifying the scene name for the plot. Default is `scene`.

Details

The wirePlot function is used to create a 3D wireframe plot that visualizes the relationship between two factors and a response variable. The plot can be customized in various ways, including changing axis labels, adding subtitles, and choosing the color palette.

Value

The function wirePlot returns an invisible list containing:

plot	The generated wireframe plot.
grid	The grid data used for plotting.

See Also

contourPlot, paretoChart.

Examples

```
# Example 1: Basic wireframe plot
x <- seq(-10, 10, length = 30)
y <- seq(-10, 10, length = 30)
z <- outer(x, y, function(a, b) sin(sqrt(a^2 + b^2)))
wirePlot(x, y, z, main = "3D Wireframe Plot", xlab = "X-Axis", ylab = "Y-Axis", zlab = "Z-Axis")

fdo = rsmDesign(k = 3, blocks = 2)
fdo$.response(data.frame(y = rnorm(fdo$nrow())))

#I - display linear fit
wirePlot(A,B,y, data = fdo, form = "linear")

#II - display full fit (i.e. effect, interactions and quadratic effects
wirePlot(A,B,y, data = fdo, form = "full")
```

```
#III - display a fit specified before
fdo$set.fits(fdo$lm(y ~ B + I(A^2)))
wirePlot(A,B,y, data = fdo, form = "fit")

#IV - display a fit given directly
wirePlot(A,B,y, data = fdo, form = "y ~ A*B + I(A^2)")

#V - display a fit using a different colorRamp
wirePlot(A,B,y, data = fdo, form = "full", col = 2)
```

`wirePlot3` *wirePlot3: function to create a ternary plot (3D wire plot)*

Description

This function creates a ternary plot for mixture designs (i.e. object of class `mixDesign`).

Usage

```
wirePlot3(
  x,
  y,
  z,
  response,
  data = NULL,
  main,
  xlab,
  ylab,
  zlab,
  form = "linear",
  col = "Rainbow",
  steps,
  plot = TRUE
)
```

Arguments

`x`	Factor 1 of the `mixDesign` object.
`y`	Factor 2 of the `mixDesign` object.
`z`	Factor 3 of the `mixDesign` object.
`response`	the response of the `mixDesign` object.
`data`	The `mixDesign` object from which x,y,z and the response are taken.
`main`	Character string specifying the main title of the plot.
`xlab`	Character string specifying the label for the x-axis.

ylab	Character string specifying the label for the y-axis.
zlab	Character string specifying the label for the z-axis.
form	A character string or a formula with the syntax ‘y ~ A + B + C‘. If form is a character string, it has to be one of the following: • ‘linear‘ • ‘quadratic‘ How the form influences the output is described in the reference listed below. By default, form is set to ‘linear‘.
col	Character string specifying the color palette to use for the plot (e.g., `Rainbow`, `Jet`, `Earth`, `Electric`). Default is `Rainbow`.
steps	A numeric value specifying the resolution of the plot, i.e., the number of rows for the square matrix, which also represents the number of grid points per factor. By default, steps is set to 25.
plot	Logical value indicating whether to display the plot. Default is TRUE.

Value

The function wirePlot3 returns an invisible matrix containing the response values as NA's and numerics.

See Also

mixDesign.c, mixDesign, contourPlot3.

Examples

```
#Example 1
mdo <- mixDesign(3, 2, center = FALSE, axial = FALSE, randomize = FALSE, replicates = c(1, 1, 2, 3))
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)
mdo$.response(elongation)
wirePlot3(A, B, C, elongation, data = mdo, form = "quadratic")

#Example 2
mdo <- mixDesign(3,2, center = FALSE, axial = FALSE, randomize = FALSE, replicates  = c(1,1,2,3))
mdo$names(c("polyethylene", "polystyrene", "polypropylene"))
mdo$units("percent")
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)
mdo$.response(elongation)
wirePlot3(A, B, C, elongation, data = mdo, form = "linear")
wirePlot3(A, B, C, elongation, data = mdo, form = "quadratic",
           col = "Jet")
wirePlot3(A, B, C, elongation, data = mdo,
           form = "elongation ~ I(A^2) - B:A + I(C^2)",
           col = "Electric")
wirePlot3(A, B, C, elongation, data = mdo, form = "quadratic",
           col = "Earth")
```

Index

adSim, 3
aliasTable, 5
as.data.frame_facDesign, 5

blocking, 6

cg, 7, *10*, *12*, *14*, *49*, *51*
cg_HistChart, *8*, 9, *12*, *14*
cg_RunChart, *8*, *10*, 11, *14*
cg_ToleranceChart, *8*, *10*, *12*, 13
code2real, 15
confounds, 15
contourPlot, 16, *117*
contourPlot3, 18, *65*, *67*, *119*

desirability, 20, *23*, *24*, *74*, *76*, *105*
desirability.c, *20*, 21, *107*
desOpt, 23, *74*
dgamma3, 24
Distr, 25, *29*, *30*, *32*, *45*
DistrCollection, *27*, 28, *32*, *45*
distribution, *27*, *30*, 31, *45*
dlnorm3, 32
doeFactor, 33
dotPlot, 35
dweibull3, 36

facDesign, *6*, *24*, 37, *46*, *47*, *63*, *65*, *72*, *73*, *76*, *79*, *80*, *101*, *104*, *109*, *110*
facDesign.c, *5*, *6*, *15*, *16*, *38*, 39, *46*, *47*, *62*, *65*, *71*, *74*, *76*, *78*, *99*, *101*, *103–105*, *107*, *115*
FitDistr, *4*, *27*, *30*, 44, *91*, *97*
fracChoose, *5*, *38*, 46, *47*, *99*, *101*
fracDesign, *5*, *38*, *46*, 46, *63*, *65*, *73*, *79*, *80*, *101*, *104*, *109*, *110*

gageLin, 48, *50*, *51*
gageLinDesign, *49*, 49
gageRR, *49*, 50, *62*
gageRR.c, *50*, *51*, 51, *62*
gageRRDesign, *51*, 61, *73*, *80*, *109*, *110*

interactionPlot, 62, *72*

mixDesign, *19*, 63, *67*, *104*, *119*
mixDesign.c, *19*, *44*, *65*, 65, *119*
MSALinearity, *49*, *50*, 68
mvPlot, 69

normalPlot, 71

oaChoose, 73
optimum, *21*, *23*, *24*, 74, *105*, *107*
overall, *21*, *23*, *74*, 75

paretoChart, *17*, 76, *117*
paretoPlot, *72*, 78
pbDesign, *38*, *47*, 80, *101*, *110*
pbDesign.c, 81
pbFactor, 83
pcr, 84
pgamma3, 87
plnorm3, 88
ppPlot, 89, *97*
print_adtest, 91
pweibull3, 92

qgamma3, 93
qlnorm3, 94
qqPlot, *91*, 95
qweibull3, 97

randomize, 98
rsmChoose, *46*, 99, *101*
rsmDesign, *38*, *46*, *47*, *65*, *73*, *76*, *80*, *99*, 100, *104*, *109*, *110*

simProc, 101
snPlot, 102
starDesign, 103
steepAscent, 105, *107*

steepAscent.c, *105*, 106
summaryFits, 107

taguchiChoose, 108
taguchiDesign, *38*, *47*, 109
taguchiDesign.c, *44*, *102*, 111
taguchiFactor, *34*, 114

wirePlot, *17*, 115
wirePlot3, *19*, *65*, *67*, 118