# Package 'DoE.base'

January 20, 2025

**Title** Full Factorials, Orthogonal Arrays and Base Utilities for DoE Packages

**Version** 1.2-4

**Depends** R (>= 2.10), grid, conf.design

**Imports** stats, utils, graphics, grDevices, vcd, combinat, MASS, lattice, numbers, partitions

**Suggests** FrF2, DoE.wrapper, RColorBrewer

**Date** 2023-10-17

**Description** Creates full factorial experimental designs and designs based on orthogonal arrays for (industrial) experiments. Provides diverse quality criteria. Provides utility functions for the class design, which is also used by other packages for designed experiments.

**License** GPL (>= 2)

**LazyLoad** yes

**LazyData** yes

**Encoding** UTF-8

**URL** https://prof.bht-berlin.de/groemping/DoE/, https://prof.bht-berlin.de/groemping/

**NeedsCompilation** no

**Maintainer** Ulrike Groemping <ulrike.groemping@bht-berlin.de>

**Author** Ulrike Groemping [aut, cre], Boyko Amarov [ctb], Hongquan Xu [ctb]

**Repository** CRAN

**Date/Publication** 2023-11-14 19:40:02 UTC

## Contents

---

`DoE.base-package` *Full factorials, orthogonal arrays and base utilities for DoE packages*

---

## Description

This package creates full factorial designs and designs from orthogonal arrays. In addition, it provides some basic utilities like an exporting function for the DoE packages FrF2, DoE.wrapper and RcmdrPlugin.DoE, and some diagnostics for general orthogonal arrays (generalized word length calculations).

## Details

The package is still in under development phase. Details about combining designs are particularly likely to be changed in the future (`param.design`, `cross.design`). Please contact me, if you have suggestions.

This package designs full factorial experiments (function `fac.design`) and experiments based on orthogonal arrays (`oa.design`). Some aspects of functions `fac.design` and `oa.design` have been modeled after the functions of the same name given in Chambers and Hastie (1993) (e.g. for the option `factor.names` or for outputting a data frame with attributes). However, S compatibility has not been considered in devising this package.

The orthogonal arrays underlying function `oa.design` are mainly taken from Kuhfeld (2009). While the arrays generally guarantee estimability of main effects in case there are no (or negligible) active interactions, some of them can also be used for designs for which some interactions are to be estimated, if only few of the design columns are used for experimentation. Optimization for such purposes and check of fitness for such purposes is supported, cf. `generalized.word.length`.

The package provides class `design` for use also by packages **FrF2**, **DoE.wrapper** and **RcmdrPlugin.DoE**. Furthermore, it provides utilities for printing, plotting, summarizing, exporting and combining experimental designs. Package **FrF2** relies on function `fac.design` for full factorials in 2-level factors.

## Acknowledgments

Thanks are due to Peter Theodor Wilrich for various useful suggestions in the early phase of this project!

## Author(s)

Ulrike Groemping

Maintainer: Ulrike Groemping <groemping@bht-berlin.de>

## References

Chambers, J.M. and Hastie, T.J. (1993). *Statistical Models in S*, Chapman and Hall, London.

Groemping, U. (2018). R Package DoE.base for Factorial Designs. *Journal of Statistical Software* **85**(5), 1–41. https://www.jstatsoft.org/issue/view/v085.

Hedayat, A.S., Sloane, N.J.A. and Stufken, J. (1999) *Orthogonal Arrays: Theory and Applications*, Springer, New York.

Kuhfeld, W. (2009). Orthogonal arrays. Website courtesy of SAS Institute https://support.sas.com/techsup/technote/ts723b.pdf and references therein.

## See Also

Functions `fac.design`, `oa.design` for generating designs, and various functions (`generalized.word.length`) for optimizing and checking a designs properties,
class `design` which is utilized also by packages **FrF2** and **DoE.wrapper**.
Furthermore, there are various utility functions like `export.design` or `add.response` and functions `cross.design` or `param.design` for combining designs.
Finally, several `methods for class design objects` are provided, especially also functions `formula.design`

and lm.design for automatic generation of linear models (but beware: these are convenience functions that provide a quick first look but NOT necessarily the best statistical approach to analysis!).

---

add.response *Function to add response values to an experimental design*

---

## Description

This function allows to add numeric response variables to an experimental plan of class design. The responses are added both to the data frame and to its desnum attribute; the response.names element of the design.info attribute is updated - the function is still experimental.

## Usage

```
add.response(design, response, rdapath=NULL, replace = FALSE,
    InDec=options("OutDec")[[1]], tol = .Machine$double.eps ^ 0.5, ...)
```

## Arguments

| | |
|---|---|
| design | a character string that gives the name of a class design object, to which responses are to be added |
| response | EITHER<br>a numeric vector, numeric matrix or data frame with at least one numeric variable (the treatment of these is explained in the details section)<br>OR<br>a character string indicating a csv file that contains the typed-in response values; after reading the csv file with the csv version indicated in the InDec argument, numeric variables from response will be added to the design as responses |
| rdapath | a character string indicating the path to a stored rda file that contains the design |
| replace | logical: TRUE implies that existing variables are overwritten in design; cf. also the details section |
| InDec | decimal separator in the external csv file; defaults to the OutDec option (viewable under options("OutDec"), and also governs whether the csv-file is read with read.csv or with read.csv: separator semicolon goes with decimal comma and triggers use of read.csv2, separator comma goes with decimal point and trigggers use of read.csv. ) |
| tol | tolerance for comparing numerical values;<br>useful for designs with numeric factors and for partial replacement of response values; the value is used in comparisons of design and response via all.equal; errors from peculiar rounding behavior of spreadsheet programs can be prevented by allowing a larger tol |
| ... | further arguments; currently not used |

## Details

If response is a data frame or a matrix, responses are assumed to be all the numeric variables that are neither factor names or block names in design (i.e. names of the factor.names element of the design.info attribute or the block.name element of that same attribute) nor column names of the run.order attribute, nor name or Name.

If design already contains columns for the response(s), NA entries of these are overwritten, if all non-NA entries coincide between design and response.

The idea behind this function is as follows: After using export.design for storing an R work space with the design object and either a csv or html file externally, Excel or some other external software is used to type in experimental information. The thus-obtained data sheet is saved as a csv-file and imported into R again (name provided in argument response, and the design object with all attached information is linked to the typed in response values using function add.response.

Alternatively, it is possible to simply type in experimental results in R, both using the R commander plugin (**RcmdrPlugin.DoE**) or simply function fix. Copy-pasting into R from Excel is per default NOT possible, which has been the reason for programming this routine.

## Value

The value is a modified version of the argument object design, which remains an object of class design with the following modifications:

- Response columns are added to the data frame
- the same response columns are added to the desnum attribute
- the response.names element of the design.info attribute is added or modified

## Author(s)

Ulrike Groemping

## See Also

See also export.design

## Examples

```
plan <- fac.design(nlevels=c(2,3,2,4))
result <- rnorm(2*3*2*4)
add.response(plan,response=result)
## direct use of rnorm() is also possible, but looks better with 48
add.response(plan,response=rnorm(48))

## Not run:
export.design(path="c:/projectA/experiments",plan)
## open exported file c:/projectA/experiments/plan.html
##        with Excel
## carry out the experiment, input data in Excel or elsewhere
##        store as csv file with the same name (or a different one, just use
##        the correct storage name later in R), after deleting
##        the legend portion to the right of the data area
```

```
##      (alternatively, input data by typing them in in R (function fix or R-commander)
  add.response(design="plan",response="c:/projectA/experiments/plan.csv",
            rdapath="c:/projectA/experiments/plan.rda")
  ## plan is the name of the design in the workspace stored in rdapath
  ## assuming only responses were typed in
  ## should work on your computer regardless of system,
  ##         if you adapt the path names accordingly

## End(Not run)
```

---

`block.catlg3` *Catalogues for blocking full factorial 2-level and 3-level designs, and lists of generating columns for regular 2- and 3-level designs.*

---

## Description

The block data frames hold Yates matrix column numbers for blocking full factorials with 2-level (up to 256 runs) and 3-level factors (up to 243 runs). The Yates lists translate these column numbers into effects.

## Usage

```
block.catlg
block.catlg3
Yates
Yates3
```

## Details

The constants documented here are used for blocking full factorial designs with function `fac.design`; `Yates` and `block.catlg` are internal here, as they have long been part of package `FrF2-package`.

The block data frames hold Yates matrix column numbers for blocking full factorials with 2-level (up to 256 runs) and 3-level factors (up to 243 runs). The Yates lists translate these column numbers into effects (see below).

Data frame `block.catlg` comes from Sun, Wu and Chen (1997). Data frame `block.catlg3` comes from Cheng and Wu (2002, up to 81 runs) and has been derived from Hinkelmann and Kempthorne (2005, Table 10.6) for 243 runs. The blocking schemes from the papers are optimal; this has not been proven for the blocking scheme for 243 runs.

`Yates` is a user-visible constant that is useful in design construction:

`Yates` is a list of design column generators in Yates order (for 4096 runs), e.g. `Yates[1:8]` is identical to

`list(1,2,c(1,2),3,c(1,3),c(2,3),c(1,2,3))`.

`Yates3` is a constant for 3-level designs, for which there are coefficients rather than generating factor numbers in the list.

### Author(s)

Ulrike Groemping

### References

Cheng, S.W. and Wu, C.F.J. (2002). Choice of Optimal Blocking Schemes in Two-Level and Three-Level Designs. *Technometrics* **44**, 269-277.

Hinkelmann, K. and Kempthorne, O. (2005). *Design and analysis of experiments, Vol.2*. Wiley, New York.

Sun, D.X., Wu, C.F.J. and Chen, Y.Y. (1997). Optimal blocking schemes for $2^n$ and $2^{n-p}$ designs. *Technometrics* **39**, 298-307.

---

`Class design and accessors`

*Class design and its accessor functions*

---

### Description

Convenience functions to quickly access and modify attributes of data frames of the class design; methods for the class are described in a separate help topic

### Usage

```
undesign(design)
redesign(design, undesigned)
desnum(design)
desnum(design) <- value
run.order(design)
run.order(design) <- value
design.info(design)
design.info(design) <- value
factor.names(design)
factor.names(design, contr.modify = TRUE, levordold = FALSE) <- value
response.names(design)
response.names(design, remove=FALSE) <- value
col.remove(design, colnames)
ord(matrix, decreasing=FALSE)
```

### Arguments

| | |
|---|---|
| `design` | data frame of S3 class `design`. For the structures of design objects, refer to the details section and to the value sections of the functions that create them. |
| `undesigned` | an object that is currently not a design but could be (e.g. obtained by applying function `undesign` |

| | |
|---|---|
| value | an appropriate replacement value:<br>a numeric version of the design matrix for function desnum (usage not encouraged for non-experts!)<br>a run order data frame for function run.order (usage not encouraged for non-experts!)<br>a list with appropriate design information for function design.info (usage not encouraged for non-experts!)<br>for function `factor.names<-` a character vector of new factor names (levels remain unchanged) or a named list of level combinations for the factors, like factor.names in function fac.design<br>for function `response.names<-` a character vector of response names referring to variables which are already available in design |
| contr.modify | logical to indicate whether contrasts are to be modified to match the new levels; relevant for R factors only, not for numeric design variables;<br>if TRUE, factors with 2 levels get -1/+1 contrasts, factors with more than two quantitative levels get polynomial contrasts with scores identical to the factor levels, and factors with more than two character levels get treatment contrasts;<br>if FALSE, the contrasts remain unchanged from their previous state.<br>If solely the contrasts are to be changed, function change.contr is preferrable. |
| levordold | logical to indicate whether the level ordering should follow the old function behavior;<br>the new behavior (from version 0.27) is more plausible, in that the level ordering in the new factor.names corresponds to the factor.names entry of the design.info attribute; previously, the automatic level ordering of factor levels deviated from that order which even led to a changed level order when reassigning exactly the factor.names element of the design.info attribute |
| remove | logical to indicate whether responses not indicated in value are to be removed from the design altogether.<br>If TRUE, the respective columns are deleted from the design. Otherwise, the columns remain in the data frame but loose their status as a response variable. |
| colnames | character vector of names of columns to be removed from the design; design factors or the block factor cannot be removed; with non-numeric variables, the desnum attribute of the design may have to be manually modified for removing the respective columns in some cases. |
| matrix | matrix, data frame or also object of class design that is to be ordered column by column |
| decreasing | logical, indicates whether decreasing order or not (increasing is default) |

## Details

Items of class design are data frames with attributes. They are generated by various functions that create experimental designs (cf. see also section), and by various utility functions for designs like the above extractor function for class design.

The data frame itself always contains the design in uncoded form. For many design generation functions, these are factors. For designs for quantitative factors (bbd, ccd, lhs, 2-level designs with center points), the design variables are numeric. This is always indicated by the design.info element quantitative, for which all components are TRUE in that case.

Generally, its attributes are `desnum`, `run.order`, and `design.info`.
Attribute `desnum` contains a numeric coded version of the design. For factor design variables, the content of `desnum` depends on the contrast information of the factors (cf. `change.contr` for modifying this).
Attribute `run.order` is a data frame with run order information (standard order, randomized order, order with replication info),
and the details of `design.info` partly depend on the type of design.

`design.info` generally is a list with first element `type`, further info on the design, and some options of the design call regarding randomization and replication. For almost all design types, elements include

**nruns** number of runs (not adjusted for replications)

**nfactors** number of factors

**factor.names** named list, as can be handed to function `oa.design`

**replications** the integer number of replications (1=unreplicated)

**repeat.only** logical indicating whether replications are only repeat runs but not truly replicated

**randomize** logical indicating whether the experiment was randomized

**seed** integer seed for the random number generator
note that the randomization behavior has changed with R version 3.6.0;
section "Warning" provides information on reproducing randomized designs.

**response.names** in the presence of response data only; the character vector identifying response columns in the data frame

**creator** contains the call or the list of menu settings within package **RcmdrPlugin.DoE** that led to creation of the design.
Note that the randomization behavior has changed with R version 3.6.0;
section "Warning" provides information on reproducing randomized designs.

For some design types, notably designs of types starting with "FrF2" and designs that have been created by combining other designs, there can be substantial additional information available from the `design.info` attribute in specialized situations. Detailed information on the structure of the `design.info` attribute can be found in the value sections of the respective functions. A tabular overview of the available `design.info` elements is given on the authors homepage.

Function `undesign` removes all design-related attributes from a class design object; this may be necessary for making some independent code work on design objects. (For example, function `reshape` from package **stats** does not work on a class design object, presumably because of the specific extractor method for class `design`.) Occasionally, one may also want to reconnect a processed undesigned object to its design properties. This is the purpose of function `redesign`.

The functions `desnum`, `run.order`, and `design.info` extract the respective attribute, i.e. e.g. function `design.info` extracts the design information for the design. The corresponding assignment functions should only be used by very experienced users, as they may mess up things badly if they are used naively .

The functions `factor.names` and `response.names` extract the respective elements of the `design.info` attribute. The corresponding assignment functions allow to change factor names and/or factor codes and to exclude or include a numeric variable from the list of responses that are recognized as such by analysis procedures. Note that the `response.names` function can (on request, not by default)

remove response variables from the data frame design. However, it is not directly able to add new responses from outside the data frame design. This is what the function add.response is for.

Function col.remove removes columns from the design and returns the design without these columns and an intact class design structure.

## Value

| | |
|---|---|
| desnum | returns a numeric matrix, the corresponding replacement function modifies a class design object |
| run.order | returns a 3-column data frame with standard and actual run order as well as a run order with replication identifiers attached; the corresponding replacement function modifies a class design object |
| design.info | returns the design.info attribute of the design; the corresponding replacement function modifies a class design object |
| factor.names | returns a named list the names of which are the names of the treatment factors of the design while the list elements are the vectors of levels for each factor |
| 'factor.names<-' | returns a class design object with modified factor names information (renamed factors and/or changed factor levels); |
| response.names | returns a character vector of response names that (names of numeric variables within the data frame design that are to be treated as response variables ) ; the corresponding replacement function modifies the design |
| 'response.names<-' | returns a class design object with modified response names information (add or remove numeric columns of the design to or from set of response variables), and potentially response columns removed from the design. |
| col.remove | returns a class design object with some columns removed from both the design itself and the desnum attribute. Response columns may be removed, but factor or block columns may not. |
| ord | returns an index vector that orders the matrix or data frame; for example, design[ord(design),] orders the design in increasing order with respect to the first, then the second etc. factor. |

## Warning

Function sample is used for the randomization functionality of this package. With R version 3.6.0, the behavior of this function has changed. Since the R version is not stored with a class design object, please check carefully if a design you want to reproduce based on a given creator or seed element of the design.info attribute has the expected randomization order.

The randomization order of a design that was created with the default settings under R version 3.6.0 or newer can only be reproduced with such a new R version.
If an R version 3.6.0 or newer is used for reproducing the randomization order of a randomized design that was created with an R version before 3.6.0, the RNGkind setting has to be modified:
RNGkind(sample.kind="Rounding")
activates the old behavior,
RNGkind(sample.kind="default")

switches back to the recommended new behavior.
For an example, see the documentation of the example data set VSGFS.

## Note

Note that R contains a few functions that generate or work with an S class design, which is cursorily documented in Appendix B of the white book (Chambers and Hastie 1993) to consist of a data frame of R factors which will later be extended by numeric response columns. Most class design objects as defined in packages **DoE.base** and **FrF2** are also compatible with this older class design; they are not, however, as soon as quantitative factors are involved, like for designs with center points in package FrF2 or for most designs in package **DoE.wrapper** (not yet on CRAN). If feasible with reasonable effort and useful, functions for the class design documented here incorporate the functions for the S class design (notably function plot.design).

This package is still under development; suggestions and bug reports are welcome.

## Author(s)

Ulrike Groemping

## References

Chambers, J.M. and Hastie, T.J. (1993). *Statistical Models in S*, Chapman and Hall, London.

## See Also

See also the following functions known to produce objects of class design: FrF2, pb, fac.design, oa.design, bbd.design, ccd.design, ccd.augment, lhs.design, as well as cross.design, param.design, and utility functions in this package for reshaping designs.
There are also special methods for class design ([.design, print.design, summary.design, plot.design)

## Examples

```
oa12 <- oa.design(nlevels=c(2,2,6))


#### Examples for factor.names and response.names
  factor.names(oa12)
  ## rename factors
  factor.names(oa12) <- c("First.Factor", "Second.Factor", "Third.Factor")
  ## rename factors and relabel levels of first two factors
  namen <- c(rep(list(c("current","new")),2),list(""))
  names(namen) <- c("First.Factor", "Second.Factor", "Third.Factor")
  factor.names(oa12) <- namen
  oa12

  ## add a few variables to oa12
  responses <- cbind(temp=sample(23:34),y1=rexp(12),y2=runif(12))
  oa12 <- add.response(oa12, responses)
  response.names(oa12)
  ## temp (for temperature) is not meant to be a response
```

```
  ## --> drop it from responselist but not from data
  response.names(oa12) <- c("y1","y2")

## looking at attributes of the design
  desnum(oa12)
  run.order(oa12)
  design.info(oa12)

## undesign and redesign
  u.oa12 <- undesign(oa12)
  str(u.oa12)
  u.oa12$new <- rnorm(12)
  r.oa12 <- redesign(oa12, u.oa12)
## make known that new is also a response
  response.names(r.oa12) <- c(response.names(r.oa12), "new")
## look at design-specific summary
  summary(r.oa12)
## look at data frame style summary instead
  summary.data.frame(r.oa12)
```

---

`contr.FrF2` *Contrasts for orthogonal Fractional Factorial 2-level designs*

---

## Description

Contrasts for orthogonal Fractional Factorial 2-level designs

## Usage

```
contr.FrF2(n, contrasts=TRUE)
```

## Arguments

| | |
|---|---|
| `n` | power of 2; number of levels of the factor for which contrasts are to be generated |
| `contrasts` | must always be `TRUE`; option needed for function `model.matrix` to work properly |

## Details

This function mainly supports `-1/+1` contrasts for 2-level factors. It does also work if the number of levels is a power of 2. For more than four levels, the levels of the factor must be in an appropriate order in order to guarantee that the columns of the model matrix for an FrF2-derived structure are orthogonal.

## Value

The function returns orthogonal contrasts for factors with number of levels a power of 2. All contrast columns consist of -1 and +1 entries (half of each). If factors in orthogonal arrays with 2-level factors are assigned these contrasts, the columns of the model matrix for the main effects model are orthogonal to each other and to the column for the intercept.

## Note

This package is currently under intensive development. Substantial changes are to be expected in the near future.

## Author(s)

Ulrike Groemping

## See Also

See Also contrasts, FrF2, fac.design, oa.design, pb

## Examples

```
## assign contr.FrF2 contrasts to a factor
status <- as.factor(rep(c("current","new"),4))
contrasts(status) <- contr.FrF2(2)
contrasts(status)
```

---

# corrPlot *Function to Visualize Correlations Between Model Matrix Columns for an Experimental Design*

---

## Description

Function corrplot plots absolute or squared values of correlations between model matrix columns of main effects up to three-factor interactions for factorial designs.

## Usage

```
corrPlot(design, scale = "corr", recode = TRUE, cor.out = TRUE, mm.out=FALSE,
    main.only = TRUE, three = FALSE, run.order=FALSE,
    frml=as.formula(ifelse(three, ifelse(run.order, "~ run.no + .^3", "~ .^3"),
                                  ifelse(run.order, "~ run.no + .^2", "~ .^2"))),
   pal = NULL, col.grid = "black", col.small = "grey", lwd.grid = 1.5, lwd.small = 0.5,
    lty.grid = 1, lty.small = 3, cex.y = 1, cex.x = 0.7, x.grid = NULL,
   main = ifelse(scale == "corr", "Plot of absolute correlations", ifelse(scale == "R2",
          "Plot of squared correlations",
          "Plot of absolute correlations of coefficient estimates")),
    split = 0, ask = (split > 0), ...)
```

## Arguments

| | |
|---|---|
| design | a class design object, or an object that can be made into that class by function data2design |
| scale | "corr" for absolute correlation of model matrix columns (default),<br>"R2" for squared correlation of model matrix columns,<br>"corr.est" for absolute correlation of coefficient estimates;<br>"corr.est" works for model matrices with full column rank only |
| recode | logical indicating whether or not to recode each column into normalized orthogonal coding with function contr.XuWu |
| cor.out | logical; if TRUE (default), the correlation matrix is invisibly returned |
| mm.out | logical; if TRUE (default: FALSE), the correlation matrix is invisibly returned, with the model matrix attached to it as an attribute |
| main.only | logical; if TRUE (default), only correlations with main effects columns are visualized, otherwise also those with two-factor interactions |
| three | logical; if FALSE (default), only two-factor interactions are included, otherwise also three-factor interactions |
| run.order | logical; if TRUE, the linear run order effect is included into the plot, and main effects are shown on the horizontal axis;<br>used in conjunction with function rerandomize.design, this option can help to find a suitable random order with reasonably low correlation to the effects of interest. |
| frml | the model formula; useful, if absolute correlation for the coefficient estimates are desired in a situation where a full model has a rank deficiency; for requirements on the formula, see the Details section. |
| pal | NULL (default), or a color palette. If NULL, the color palette depends on whether or not package **RColorBrewer** is available: if so, the Blues palette with nine shades is used; otherwise, a 10 level palette of heat colors augmented with white is used. The number of colors in pal determines the number of bins for plotting. |
| col.grid | color of the main grid lines |
| col.small | color of the small grid lines |
| lwd.grid | width of the main grid lines |
| lwd.small | with of the small grid lines |
| lty.grid | line type of the main grid lines |
| lty.small | line type of the small grid lines |
| cex.y | size of tick mark labels on vertical axis |
| cex.x | size of tick mark labels on horizontal axis |
| x.grid | vector of numerical positions for thicker vertical grid lines (default: NULL for none) |
| main | title |
| split | an integer number (default: 0, no split) of columns after which to split the horizontal axis; if this number is nonzero and smaller than the number of columns to display, several plots are created; note: the color legend needs attention, since it may differ between the different plots, depending on the plot's range of values |

| | |
|---|---|
| ask | logical; if yes (default in case of splitting, otherwise not), the user is asked to confirm creation of each new plot |
| ... | additional arguments to function levelplot |

## Details

The function can be used for visualizing confounding within an experimental design. It is strongly recommended to apply it to designs with columns coded in normalized orthogonal coding (contr.XuWu, contr.XuWuPoly, if applicable also contr.FrF2). Nevertheless, for factors with more than two levels, the picture shown depends on the choice of normalized orthogonal coding (see examples). Option recode=FALSE is there to allow to keep a suitably-chosen normalized orthogonal coding for each factor.

The function shows the absolute correlation or squared correlation between model matrix columns, or, on request and if possible, the absolute correlation between estimated coefficients (other than the intercept). In case the latter cannot be obtained for the full model, a model formula can be specified with option frml. Note that it is implicitly assumed that all main effects are included in the model formula, and for main.only=FALSE also all two-factor interactions.

For resolution III and higher designs, the vertical axis shows the main effects (and, if main.only=FALSE, also the two-factor interactions), the horizontal axis shows the two-factor interactions (and, if three=TRUE, also the three-factor interactions). For resolution II designs, the horizontal axis additionally shows the main effect columns (since they are correlated with other main effect columns). For resolution VI and higher designs, the function stops with an error. For resolution V designs, the function shows correlations between two-factor interactions on the vertical axis and three-factor interactions on the horizontal axis, if both are activated. The most interesting cases are designs of resolution up to IV.

The diagonal of the correlation matrix is set to NA before plotting, in order to be able to better see differences in case there are only relatively low correlations.

With scale="R2", and using normalized orthogonal coding, some sums of matrix entries coincide with contributions to generalized word counts (resolution II: main effects with main effects; resolution III: main effects with two-factor interactions; resolution IV: main effects with three-factor interactions; see Groemping and Xu (2014) for the background of this result and Groemping (2017).

## Value

The entire matrix of absolute correlations is output invisibly.

## Author(s)

Ulrike Groemping, Berliner Hochschule fuer Technik

## References

Groemping, U. (2017). Frequency Tables for the coding invariant quality assessment of factorial designs. *IISE Transactions* **49**(5), 505–517.

Groemping, U. and Xu, H. (2014). Generalized resolution for orthogonal arrays. *The Annals of Statistics* **42**, 918–939.

The function works similarly to `colormap` in package **daewr** (but offers significantly more choices). That package accompanies the following book:

Lawson, J. (2013). Design and Analysis of Experiments with R. CRC, Boca Raton.

## See Also

See Also as `levelplot`, ~~~

## Examples

```
## this is with the default contr.XuWu recoding
mat <- corrPlot(VSGFS)
round(mat, 2)

## NOT RECOMMENDED: force-keep non-normalized coding
corrPlot(VSGFS, recode=FALSE)  # not useful!

## custom normalized orthogonal coding
## that has correlations more concentrated on fewer columns
plan <- change.contr(VSGFS, "contr.XuWuPoly")
contrasts(plan$CDs) <- contr.FrF2(4)
corrPlot(plan, recode=FALSE)  # that is the purpose of recode=FALSE

corrPlot(VSGFS, main.only=FALSE, three=TRUE, cex.x=0.5, cex.y=0.5, split=100)
```

---

`cross.design` *Function to cross several designs*

---

## Description

This function generates cartesian products of two or more experimental designs.

## Usage

```
cross.design(design1, design2, ..., randomize = TRUE, seed=NULL)
```

## Arguments

| | |
|---|---|
| `design1` | a data frame of class `design` that restricted by certain criteria (cf. details)<br>if `design1` is not of class `design`, crossing will nevertheless work, but the output object will be a data frame only without any design information; there is no guaranteed support for this usage |
| `design2` | a data frame of class `design` with the same restrictions for design type as for `design1`;<br>can also be a vector if `...` is not used;<br>cf. details for what is allowed regarding replications |

`...` optional further data frames that are to be crossed; they must be of class `design` with the above-mentioned restrictions for design types; the last element can also be a vector

`randomize` logical indicating whether randomization should take place after crossing the designs

`seed` integer seed for the random number generator
In R version 3.6.0 and later, the default behavior of function `sample` has changed. If you work in a new (i.e., >= 3.6.-0) R version and want to reproduce a randomized design from an earlier R version (before 3.6.0), you have to change the RNGkind setting by
`RNGkind(sample.kind="Rounding")`
before running function `cross.design`.
It is recommended to change the setting back to the new recommended way afterwards:
`RNGkind(sample.kind="default")`
For an example, see the documentation of the example data set `VSGFS`.

## Details

Crossing is carried out recursively, following the `direct.sum` approach from package **conf.design**. All but the last designs must fulfill various criteria (cf. below). The last design to be crossed can also be a vector.

Designs to be crossed must not be a blocked, nor splitplot, nor crossed, folded or Taguchi parameter design, nor designs in wide format. Furthermore, designs must not contain responses (checked via the response.names element of `design.info`).

If replications are desired, it is recommended to accomodate them in the last design. Only the last design may have `repeat.only` replications. If the last design has `repeat.only` replications and there are also proper replications in earlier designs, a warning is thrown, but the `repeat.only` replications are nevertheless accomodated; this is experimental and may not yield the expected results under all circumstances.

## Value

Function `cross.design` returns a simple data frame without design information, if `design1` is not of class `design`.

Otherwise, the value is a data frame of class `design` with type "crossed" and the following extraordinary elements:

`cross.nruns` vector of run numbers of individual designs

`cross.nfactors` vector of numbers of factors of individual designs

`cross.types` vector of types of individual designs

`cross.randomize`
vector of logicals (randomized or not) of individual desigs

`cross.seed` vector of seeds of individual designs

`cross.replications`
vector of numbers of replications of individual designs

`cross.repeat.only`
: vector of logicals (repeat.only or not) of individual designs

`cross.map`
: list with the map vectors for component designs of type `FrF2.estimable`

`cross.selected.columns`
: `NULL` (if no `oa` type design) or list of column vectors for each design

`cross.nlevels`
: list with the `nlevels` vectors for those component designs that have them

The standard elements are as usual, with `randomize` and `seed` referring to the randomization within function `cross.design` itself (previous randomizations are shown under `cross.randomize` and `cross.seed`).

The `nlevels` element of `design.info` is available only if it is available for all designs that have been crossed (otherwise refer to the element `cross.nlevels`.

The `creator` element of the `design.info` attribute consists is a 2-element list containing
the list `original` of all the original creators and
the element `modify` that contains the call to `cross.design`.

If present, the `clear`, `ncube`, `ncenter`, `residual.df`, `origin`, `comment`, `generating.oa` elements of `design.info` are vector-valued.

If present, the `generators` element of `design.info` is a list of character vectors.

If present, the `aliased` and `catlg.entry` elements of `design.info` are lists of lists.

## Warning

Since R version 3.6.0, the behavior of function `sample` has changed (correction of a biased previous behavior that should not be relevant for the randomization of designs). For reproducing a randomized design that was produced with an earlier R version, please follow the steps described with the argument `seed`.

## Note

This function is still experimental.

## Author(s)

Ulrike Groemping

## See Also

See Also `param.design`

## Examples

```
## creating a Taguchi-style inner-outer array design
## with proper randomization
##   function param.design would generate such a design with all outer array runs
##     for each inner array run conducted in sequence
##   alternatively, a split-plot approach can also handle control and noise factor
##     designs without necessarily crossing two separate designs
des.control <- oa.design(ID=L18)
```

```
des.noise <- oa.design(ID=L4.2.3,nlevels=2,factor.names=c("N1","N2","N3"))
crossed <- cross.design(des.control, des.noise)
crossed
summary(crossed)
```

---

`expansive.replace` *Expansive replacement for two orthogonal arrays*

---

## Description

Expansive replacement for two orthogonal arrays

## Usage

```
expansive.replace(array1, array2, fac1 = NULL, all = FALSE)
```

## Arguments

| | |
|---|---|
| `array1` | an orthogonal array, must be a matrix;<br>the levels of column `fac1` of this array are replaced by the corresponding runs of `array2`; they must be numbered with integers starting with 1 |
| `array2` | an orthogonal array, must be a matrix<br>this array is replaced for a column in `array1` |
| `fac1` | if `NULL`, the first suitable column of `array1` is replaced with `array2`;<br>alternatively, a suitable column number for `array1` can be specified |
| `all` | logical; `TRUE` is permitted, if `array2` is a full factorial in two factors; if `TRUE`, a list of permutations of the replacement array `array2` is used for creating all potentially combinatorially different outcomes, instead of a single expansive replacement based on the order of `array2` as stated (see Details section) |

## Details

This function mainly is meant for combining two orthogonal arrays via what Kuhfeld (2009) calls expansive replacement.

If `array2` is a full factorial in two factors, argument `all = TRUE` creates a list of expanded arrays obtained by permuting the second array in all ways that may lead to combinatorially different end results. With $s_1$ and $s_2$ the numbers of levels of the factors in `array2`, this is the number of partitions of the runs of `array2` into $s_1$ equally-sized groups, multiplied with ${s_2!}^{s_1-1}$ for the possibilities of permuting the levels of the second factor within all but the first level of the first factor. This functionality is primarily meant for the creation of strength 3 arrays in combination with arrays listed in the catalogue `oacat3` (see an example on expanding the 6-level factor in `L96.2.5.4.2.6.1`).

## Value

The function returns an object of classes `oa` and `matrix`, which can be used in function `oa.design`, or a list of such arrays, in case `all = TRUE`.

Whether or not the object is an orthogonal array depends on the choice of suitable input arrays by the user. The properties of the resulting array(s) can e.g. be inspected with functions `GWLP` or `GRind`.

## Note

This package is still under development. Bug reports and feature requests are welcome.

## Author(s)

Ulrike Groemping

## References

Kuhfeld, W. (2009). Orthogonal arrays. Website courtesy of SAS Institute https://support.sas.com/techsup/technote/ts723b.pdf and references therein.

## See Also

See Also oacat, oacat3

## Examples

```
myL24.2.14.6.1 <- expansive.replace(L24.2.12.12.1, L12.2.2.6.1)

L96.2.6.3.1.4.2_list <- expansive.replace(L96.2.5.4.2.6.1,
      cbind(U=rep(c(1,2),each=3), V=rep(1:3,2)), all=TRUE)
## the list of 60 resolution IV arrays can be used for design creation,
## e.g. as follows:
## Not run:
     ## resolution IV designs obtained from the 60 different arrays
     deslist <- lapply(L96.2.6.3.1.4.2_list,
         function(aa) oa.design(aa, nlevels=c(2,2,2,2,3,4,4), columns="min34"))
     table(A4s <- sapply(deslist, length4))  ## a single best design exists
     best <- deslist[[which(A4s < 2)]]
     GWLP(best)

## End(Not run)
```

---

# export.design *Function for exporting a design object*

---

## Description

Function for exporting a design object

## Usage

```
export.design(design, response.names = NULL,
       path = ".", filename = NULL, legend = NULL, type = "html",
       OutDec = options("OutDec")$OutDec, replace = FALSE, version = 2, ...)
html(object, ...)
## S3 method for class 'data.frame'
```

```
html(object, file = paste(first.word(deparse(substitute(object))),
   "html", sep = "."), append = FALSE, link = NULL, linkCol = 1, bgs.col = NULL,
         OutDec=options("OutDec")$OutDec, linkType = c("href", "name"), ...)
```

## Arguments

| | |
|---|---|
| design | A data frame of class design; it must be stored in the global environment and referred to by its name, i.e. it cannot be created "on the fly". |
| response.names | if not NULL (default), this must be a character vector of response names; the exported file contains a column for each entry; it is NOT necessary to include responses that are already present in the design object! |
| path | the path to the directory where the export files are to be stored; the default corresponds to the R working directory that can (on some systems) be looked at using getwd() |
| filename | character string that gives the file name (without extension) for the files to be exported; if NULL, it is the name of the design object |
| legend | data frame containing legend information; if NULL, the legend is automatically generated from the factor.names element of design.info(design) |
| type | one of "rda", "html", "csv", or "all". An R workspace with just the design object is always stored as an "rda" object. If one of the other types is specified, the design is additionally exported to "html" or "csv" or both. The "csv" file contains the design itself only, with formatting depending on the OutDec option. The "html" file contains some additional legend information and row color formatting. |
| OutDec | decimal separator for the output file; one of "." or ","; the default is the option setting in the R options; this option also directs whether write.csv or write.csv2 is used and is very important for usability of the exported files e.g. with Excel |
| replace | logical indicating whether an existing file should be replaced; if FALSE (default), the routine aborts without any action if one of the files to be created exists; checking is not case-sensitive in order to protect users on case-insensitive platforms from inadvertent replacing of files (i.e. you cannot have TEST.html and test.html, even if it were allowed on your platform) |
| version | the save version for the rda file; starting with R 3.6.0, the default save version is 3; that version cannot be read by R versions before 3.5.0. Therefore, the default of this package is still 2, but can be overwritten by users who are certain to use the file only on new R versions. |
| object | object to be exported to html |
| file | file to export the object to |
| append | append data frame to existing file ? |
| link | not used, unchanged from package **Hmisc** |
| linkCol | not used, unchanged from package **Hmisc** |
| bgs.col | background colors for data frame rows, default white and grey |
| linkType | not used, unchanged from package **Hmisc** |
| ... | further arguments to function html, usable e.g. for modifying row coloring |

## Details

Function `export.design` always stores an R workspace that contains just the design (with attached attributes, cf. class `design`). This file is stored with ending `rda`.
If requested by options `type="csv"`, `type="html"`, or `type="all"`, `export.design` additionally creates an exported version of the design that is usable outside of R. This is achieved via functions `write.csv`, `write.csv2` or `html`. The csv-file contains the data frame itself only, the html file contains the data frame followed by the legend to the right of the data frame. The html file uses row coloring in order to prevent mistakes in recording of experimental results by mix-ups of rows. If the OutDec option is correct for the current computer, the csv and html files can be opened in Excel, and decimal numbers are correctly interpreted.

Generation of the html-file is particularly important for Taguchi inner/outer array designs in wide format, because it provides the legend to the suffix numbers of response columns in terms of outer array experimental setups!

The function `html` and its data frame method are internal.

## Value

The functions are used for their side effects and do not generate a result.

## Note

This package is currently under intensive development. Substantial changes are to be expected in the near future.

## Author(s)

Ulrike Groemping; the html functions have been adapted from package Hmisc

## References

Hedayat, A.S., Sloane, N.J.A. and Stufken, J. (1999) *Orthogonal Arrays: Theory and Applications*, Springer, New York.

## See Also

See also `FrF2-package`, `DoE.wrapper-package`

## Examples

```
  ## six 2-level factors
  test <- oa.design(nlevels=c(2,3,3,3))
  ## export an html file with legend and two responses
  ## files test.rda and test.html will be written to the current working directory,
  ##     if they do not exist yet
  ## Not run:
  export.design(test, response.names=c("pressure", "temperature"))

## End(Not run)
```

`fac.design` *Function for full factorial designs*

## Description

Function for creating full factorial designs with arbitrary numbers of levels, and potentially with blocking

## Usage

```
fac.design(nlevels=NULL, nfactors=NULL, factor.names = NULL,
        replications=1, repeat.only = FALSE, randomize=TRUE, seed=NULL,
        blocks=1, block.gen=NULL, block.name="Blocks", bbreps=replications,
        wbreps=1, block.old.behavior=FALSE)
```

## Arguments

| | |
|---|---|
| `nlevels` | number(s) of levels, vector with `nfactors` entries or single number; can be omitted, if obvious from `factor.names` |
| `nfactors` | number of factors, can be omitted if obvious from entries `nlevels` or `factor.names` |
| `factor.names` | if `nlevels` is given, `factor.names` can be a character vector of factor names. In this case, default factor levels are the numbers from 1 to the number of levels for each factor.<br>Otherwise it must be a list of vectors with factor levels. If the list is named, list names represent factor names, otherwise default factor names are used. Default factor names are the first elements of the character vector `Letters`, or the factors position numbers preceded by capital F in case of more than 50 factors. If both `nlevels` and factor.names are given, they must be compatible. |
| `replications` | positive integer number. Default 1 (i.e. each row just once). If larger, each design run is executed replication times. If `repeat.only`, repeated measurements are carried out directly in sequence, i.e. no true replication takes place, and all the repeat runs are conducted together. It is likely that the error variation generated by such a procedure will be too small, so that average values should be analyzed for an unreplicated design.<br>Otherwise (default), the full experiment is first carried out once, then for the second replication and so forth. In case of randomization, each such blocks is randomized separately. In this case, replication variance is more likely suitable for usage as error variance (unless e.g. the same parts are used for replication runs although build variation is important). |
| `repeat.only` | logical, relevant only if replications > 1. If `TRUE`, replications of each run are grouped together (repeated measurement rather than true replication). The default is `repeat.only=FALSE`, i.e. the complete experiment is conducted in `replications` blocks, and each run occurs in each block. |
| `randomize` | logical. If `TRUE`, the design is randomized. This is the default. In case of replications, the nature of randomization depends on the setting of option `repeat.only`. |

`seed` integer seed for the random number generator
In R version 3.6.0 and later, the default behavior of function `sample` has changed. If you work in a new (i.e., >= 3.6.-0) R version and want to reproduce a randomized design from an earlier R version (before 3.6.0), you have to change the RNGkind setting by
`RNGkind(sample.kind="Rounding")`
before running function `fac.design`.
It is recommended to change the setting back to the new recommended way afterwards:
`RNGkind(sample.kind="default")`
For an example, see the documentation of the example data set `VSGFS`.

`blocks` is the number of blocks into which the experiment is to be subdivided;
it must be a prime or a product of prime numbers which occur as common divisors of the numbers of levels of several factors (cf. Details section).

If the experiment is randomized, randomization happens within blocks.

`block.gen` provides block generating information.
Only specify `block.gen`, if `blocks`>1.
If `blocks` is a prime or a power of 2 (up to 2^8) or 3 (up to 3^5) or a product of powers of 2, 3, and an individual other prime, `block.gen` is not needed (but can be optionally specified).
If given, `block.gen` can be
a numeric vector of integer numbers that will be treated as a one-row matrix
OR a numeric matrix with integer elements.
There must be a row for each prime number into which `blocks` factorizes, and a column for each (pseudo)factor into which the experimental design factors can be partitioned (cf. Details and Examples sections and function `factorize`).
Rows for a p-level contributor to the block factor (p a prime) consist of entries 0 to p-1 only.

`block.name` name of the block factor, default "Blocks"

`bbreps` between block replications; these are always taken as genuine replications, not repeat runs; default: equal to `replications`; CAUTION: you should not modify `bbreps` if you do not work with blocks, because the program code uses it instead of `replications` in some places

`wbreps` within block replications; whether or not these are taken as genuine replications depends on the setting of `repeat.only`

`block.old.behavior`
logical that can be used to activate the old (prior to version 0.27) behavior of blocking full factorial designs; the new behavior is the default, as it often creates designs with less severe confounding

## Details

`fac.design` creates full factorial designs, i.e. the number of runs is the product of all numbers of levels.

It is possible to subdivide the design into blocks (one hierarchy level only) by specifying an appropriate number of blocks. The method used is a generalization of the one implemented in function `conf.design` for symmetric factorials (i.e. factorials with all factors at the same prime number of levels) and related to the method described in Collings (1984, 1989); function `conf.set` from package **conf.design** is used for checking the confounding consequences of blocking.

Note that the number of blocks must be compatible with the factor levels; it must factor into primes that occur with high enough frequency among the pseudo-factors of the design. This statement is now explained by an example: Consider a design with five factors at 2, 2, 3, 3, 6 levels. The 6-level factor can be thought of as consisting of two pseudo-factors, a 2-level and a 3-level pseudo-factor, according to the factorization of the number 6 into the two primes 2 and 3. It is possible
to obtain two blocks by confounding the two-factor interaction of the two 2-level factors and the 2-level pseudo-factor of the 6-level factor,
or to obtain three blocks by confounding the blocking factor with the three-factor interaction of the two three-level factors and the three-level pseudo-factor of the 6-level factor,
or to get six blocks, by doing both simultaneously.
It is also possible to obtain 4 or 9 or even 36 blocks, if one is happy to confound two-factor interactions with blocks. The 36 blocks are the product of the 4 blocks from the 2-level portion with the nine blocks from the 3-level portion. For each portion separately, there is a lookup-table for blocking possibilities (`block.catlg`), for up to 128 blocks in 256 runs, or up to 81 blocks in 243 runs.

5 blocks cannot be done for the above example design. Even if there were one additional factor at 5 levels, it would still not be possible to do a number of blocks with divisor 5, because this would confound the main effect of a factor with blocks and would thus generate an error.

For any primes apart from 2 or 3, only one at a time can be handled automatically. For example, if a design has three 5-level factors, it can be automatically subdivided into 5 blocks by the option `blocks=5`. It is also possible to run the design in 25 blocks; however, as 25=5*5, this cannot be done automatically but has to be requested by specifying the `block.gen` option in addition to the `blocks` option (in this case, `block.gen=rbind(c(1,0,1),c(1,1,0))` would do the job).

## Value

`fac.design` returns a data frame of S3 class `design` with attributes attached.

The experimental factors are all stored as R factors.
For factors with 2 levels, `contr.FrF2` contrasts (-1 / +1) are used.
For factors with more than 2 numerical levels, polynomial contrasts are used (i.e. analyses will per default use orthogonal polynomials).
For factors with more than 2 categorical levels, the default contrasts are used.

For changing the contrasts, use function `change.contr`.

The `design.info` attribute of the data frame has the following elements:

**type** character string "full factorial" or "full factorial.blocked"

**nruns** number of runs (replications are not counted)

**nfactors** number of factors

**nlevels** vector with number of levels for each factor

**factor.names** list named with (treatment) factor names and containing as entries vectors with coded factor levels

**nblocks** for designs of type `full factorial.blocked` only;
number of blocks

**block.gen** for designs of type `full factorial.blocked` only;
matrix the rows of which are the coefficients of the linear combinations that create block columns from of pseudo factors

**blocksize** for designs of type `full factorial.blocked` only;
size of each block (without consideration of `wbreps`)

**replication** option setting in call to `FrF2`

**repeat.only** option setting in call to `FrF2`

**bbreps** for designs of type `FrF2.blocked` only; number of between block replications

**wbreps** for designs of type `FrF2.blocked` only; number of within block replications;
`repeat.only` indicates whether these are replications or repetitions only

**randomize** option setting in call to `FrF2`

**seed** option setting in call to `FrF2`

**creator** call to function FrF2 (or stored menu settings, if the function has been called via the R commander plugin **RcmdrPlugin.DoE**)

## Warning

Since R version 3.6.0, the behavior of function `sample` has changed (correction of a biased previous behavior that should not be relevant for the randomization of designs). For reproducing a randomized design that was produced with an earlier R version, please follow the steps described with the argument `seed`.

## Note

This package is still under development. Suggestions and bug reports are welcome.

## Author(s)

Ulrike Groemping

## References

Collings, B.J. (1984). Generating the intrablock and interblock subgroups for confounding in general factorial experiments. *Annals of Statistics* **12**, 1500–1509.

Collings, B.J. (1989). Quick confounding. *Technometrics* **31**, 107–110.

## See Also

See also `FrF2`, `oa.design`, `pb`, `conf.set`, `block.catlg`

## Examples

```
## only specify level combination
fac.design(nlevels=c(4,3,3,2))
## design requested via factor.names
fac.design(factor.names=list(one=c("a","b","c"), two=c(125,275),
   three=c("old","new"), four=c(-1,1), five=c("min","medium","max")))
## design requested via character factor.names and nlevels
##    (with a little German lesson for one two three)
fac.design(factor.names=c("eins","zwei","drei"),nlevels=c(2,3,2))

### blocking designs
fac.design(nlevels=c(2,2,3,3,6), blocks=6, seed=12345)
## the same design, now unnecessarily constructed via option block.gen
## preparation: look at the numbers of levels of pseudo factors
## (in this order)
unlist(factorize(c(2,2,3,3,6)))
## or, for more annotation, factorize the unblocked design
factorize(fac.design(nlevels=c(2,2,3,3,6)))
## positions 1 2 5 are 2-level pseudo factors
## positions 3 4 6 are 4-level pseudo factors
## blocking with highest possible interactions
G <- rbind(two=c(1,1,0,0,1,0),three=c(0,0,1,1,0,1))
plan.6blocks <- fac.design(nlevels=c(2,2,3,3,6), blocks=6, block.gen=G, seed=12345)
plan.6blocks

## two blocks, default design, but unnecessarily constructed via block.gen
fac.design(nlevels=c(2,2,3,3,6), blocks=2, block.gen=c(1,1,0,0,1,0), seed=12345)

## three blocks, default design, but unnecessarily constructed via block.gen
fac.design(nlevels=c(2,2,3,3,6), blocks=3, block.gen=c(0,0,1,1,0,1), seed=12345)

## nine blocks
## confounding two-factor interactions cannot be avoided
## there are warnings to that effect
G <- rbind(CD=c(0,0,1,1,0,0),CE2=c(0,0,1,0,0,1))
plan.9blocks <- fac.design(nlevels=c(2,2,3,3,6), blocks=9, block.gen=G, seed=12345)

## further automatic designs, not run for shortening run time
## Not run:
fac.design(nlevels=c(2,2,3,3,6), blocks=4, seed=12345)
fac.design(nlevels=c(2,2,3,3,6), blocks=9, seed=12345)
fac.design(nlevels=c(2,2,3,3,6), blocks=36, seed=12345)
fac.design(nlevels=c(3,5,6,10), blocks=15, seed=12345)

## End(Not run)

## independently check aliasing
## model with block main effects and all two-factor interactions
## 6 factors: not aliased
summary(plan.6blocks)
alias(lm(1:nrow(plan.6blocks)~Blocks+(A+B+C+D+E)^2,plan.6blocks))
## 9 factors: aliased
```

```
summary(plan.9blocks)
alias(lm(1:nrow(plan.9blocks)~Blocks+(A+B+C+D+E)^2,plan.9blocks))
```

---

`factorize` *Factorize integer numbers and factors*

---

## Description

Methods to factorize integer numbers into primes or factors into pseudo factors with integer numbers of levels

## Usage

```
## S3 method for class 'factor'
factorize(x, name = deparse(substitute(x)), extension = letters,
    drop = FALSE, sep = "", ...)
## S3 method for class 'design'
factorize(x, extension = letters, sep = ".", long=FALSE, ...)
## S3 method for class 'data.frame'
factorize(x, extension = letters, sep = ".", long=FALSE, ...)
```

## Arguments

| | |
|---|---|
| `x` | factor<br>OR data frame of class `design` OR data frame |
| `name` | name to use for prefixing the pseudo factors |
| `extension` | extensions to use for postfixing the pseudo factors |
| `drop` | TRUE: have a vector only in case of just one pseudo factor |
| `sep` | separation between name and postfix for pseudo factors |
| `long` | TRUE: create a complete matrix of pseudofactors; FALSE: only create the named numbers of levels |
| `...` | currently not used |

## Details

These functions are used for blocking full factorials. The method for class `factors` is a modification of the analogous method from package **conf.design**, the other two are convenience versions for designs and data frames.

## Value

All three methods return a matrix of pseudo factors (in case `long=TRUE`) or a named numeric vector of numbers of levels of the pseudo factors (for the default `long=FALSE`).

### Note

There may be conflicts with functions from packages **conf.design** or **sfsmisc**.

### Author(s)

Ulrike Groemping; Bill Venables authored the original of factorize.factor.

### See Also

The function `factorize` from package **conf.design**,
the function `factorize` from package **sfsmisc** (no link provided, in order to avoid having to include **sfsmisc** in Suggests).

### Examples

```
factorize(12)
factorize(c(2,2,3,3,6))
factorize(fac.design(nlevels=c(2,2,3,3,6)))
unlist(factorize(c(2,2,3,3,6)))
factorize(undesign(fac.design(nlevels=c(2,2,3,3,6))))
```

---

## formula.design *Function to change the default formula for a data frame of class design to involve the correct factors with the desired effects and responses*

---

### Description

This function provides a reasonable default formula for linear model analyses of class design objects with response(s). Per default, the resulting formula refers to the first response in the design and is of design-type specific nature.

### Usage

```
## S3 method for class 'design'
formula(x, ..., response=NULL, degree=NULL, FUN=NULL,
      use.center=NULL, use.star=NULL, use.dummies=FALSE)
```

### Arguments

| | |
|---|---|
| `x` | an object of class `design` |
| `...` | further arguments to function `formula` |
| `response` | character string giving the name of the response variable (must be among the numeric columns from `x`)<br>OR<br>integer number giving the position of the response in element `response.names` of attribute `design.info` |

| | |
|---|---|
| degree | degree of the model (1=main effects only, 2=with 2-factor interactions and quadratic effects, 3=with 3-factor interactions and up to cubic effects, ... |
| FUN | function for the aggregate.design method; this must be an unquoted function name or NULL;<br>This option is relevant for repeated measurement designs and parameter designs in long format only |
| use.center | NULL or logical indicating whether center points are to be used + in the analysis; if NULL, the default is FALSE for pb and FrF2 designs with center points and TRUE for ccd designs; the option is irrelevant for all other design types. |
| use.star | NULL or logical indicating whether the star portion of a CCD design is to be used in the analysis (ignored for all other types of designs). |
| use.dummies | logical indicating whether the error dummies of a Plackett Burman design are to be used in the formula (ignored for all other types of designs). |

## Details

Function formula creates an appropriate formula for many kinds of objects, e.g. for data frames (try e.g. formula(swiss)). Function as.formula uses function formula, but cannot take any additional arguments.

The method for class design objects modifies the way a data frame would normally be treated by the formula function. This also carries through to default linear models.

Without the additional arguments, the function creates the formula with the first response from the response.names element of the design.info attribute. The default degree depends on the type of design: it is

- 1 for oa and pb
- 2 for all other design types

degree does not have an effect for response surface designs (types bbd, bbd.blocked and ccd) and latin hypercube designs (type lhs), where the function always creates the formula for a full second order model including quadratic effects.

Where degree does have an effect, it is the exponent of the sum of all experimental factors, i.e. it refers to the degree of interactions, not to powers of the variables themselves (e.g. (A+B+C)^2 for degree 2).

For designs with a block variable (types FrF2.blocked, bbd.blocked and ccd) the block variable enters the formula as a main effect factor without any interactions.

For 2-level designs with center points (types FrF2.center or pb.center), the formula contains an indicator variable center for the center points that can is used for checking whether quadratic effects are needed.

For designs with repeated measurements (repeat.only and parameter designs, the default is to analyse aggregated responses. For more detail, see the documentation of lm.design.

For optimal designs, the formula is the model formula used in optimizing the design.

## Value

a formula

### Author(s)

Ulrike Groemping

### See Also

See also formula and lm.design

### Examples

```
## indirect usage via function lm.design is much more interesting
## cf help for lm design!

 my.L18 <- oa.design(ID=L18,
     factor.names = c("one","two","three","four","five","six","seven"),
     nlevels=c(3,3,3,2,3,3,3))
 y <- rnorm(18)
 my.L18 <- add.response(my.L18, y)
 formula(my.L18)
 lm(my.L18)
```

---

## genChild — *Internal utility functions to support automatic creation of child arrays from entries of the data frame oacat*

---

### Description

The functions are used internally for creating the child arrays listed in data frame oacat from the parent arrays that come with **DoE.base** (or using full factorials).

### Usage

```
parseArrayLine(array.line)
genChild(array.list)
getArray(nbRuns, descr)
symb2oa(nbRuns, descr)
oa2symb(name)
```

### Arguments

| | |
|---|---|
| array.line | a row from data frame oacat |
| array.list | the output from function parseArrayLine |
| nbRuns | the number of runs of the array to be received |
| descr | a character string containing the description of the array to be retrieved, of the form nl1~fl1;nl2~fl2; ..., where nl stands for the number of levels and fl for their respective frequency; the string may (but need not) contain a trailing semicolon |
| name | name of an array according to the naming conventions in oacat |

## Details

Function parseArrayLine transforms information from a row of oacat into a list format digestible by function genChild.

Function genChild creates a child array from the appropriate information provided by function parseArrayLine.

Function getArray retrieves a stored orthogonal array based on the list information it receives.

Functions symb2oa and oa2symb can be used for switching between design names from data frame oacat and list type information used by functions internally. Note that the result from oa2symb is not sufficient to get back to the oa representation, but needs the number of runs in addition.

## Value

parseArrayLine returns a list with design and lineage description in symbolic form.

genChild and getArray return an array matrix of class oa.

symb2oa and oa2symb return a character string.

## Author(s)

Boyko Amarov and Ulrike Groemping

## See Also

See also oacat, oa

---

# generalized.word.length

*Functions for calculating the generalized word length pattern, projection frequency tables or optimizing column selection within an array*

---

## Description

Functions length2, length3, length4 and length5 calculate the numbers of generalized words of lengths 2, 3, 4, and 5 respectively, lengths calculates them all. Functions P3.3 and P4.4 calculate projection frequency tables, functions oa.min3, oa.min4, oa.min34, oa.maxGR (deprecated), oa.minRelProjAberr, oa.max3 and oa.max4 determine column allocations with minimum or maximum aliasing. Function nchoosek is an auxiliary function for calculating all subsets without replacement.

## Usage

```
length2(design, with.blocks = FALSE, J = FALSE)
length3(design, with.blocks = FALSE, J = FALSE, rela = FALSE)
length4(design, with.blocks = FALSE, separate = FALSE, J = FALSE, rela = FALSE)
length5(design, with.blocks = FALSE, J = FALSE, rela = FALSE)
lengths(design, ...)
```

```
## Default S3 method:
lengths(design, ...)
## S3 method for class 'design'
lengths(design, ...)
## S3 method for class 'matrix'
lengths(design, ...)
contr.XuWu(n, contrasts=TRUE)
contr.XuWuPoly(n, contrasts=TRUE)
oa.min3(ID, nlevels, all, rela = FALSE, variants = NULL, crit = "total")
oa.min4(ID, nlevels, all, rela = FALSE, variants = NULL, crit = "total")
oa.min34(ID, nlevels, variants = NULL, min3=NULL, all = FALSE, rela = FALSE)
oa.max3(ID, nlevels, rela = FALSE)
oa.max4(ID, nlevels, rela = FALSE)
oa.maxGR(ID, nlevels, variants = NULL)
oa.minRelProjAberr(ID, nlevels, maxGR = NULL)
P2.2(ID, digits = 4, rela=FALSE, parft=FALSE, parftdf=FALSE, detailed=FALSE)
P3.3(ID, digits = 4, rela=FALSE, parft=FALSE, parftdf=FALSE, detailed=FALSE)
P4.4(ID, digits = 4, rela=FALSE, parft=FALSE, parftdf=FALSE, detailed=FALSE)
nchoosek(n, k)
tupleSel(design, type="complete", selprop=0.25, ...)
## S3 method for class 'design'
tupleSel(design, type="complete", selprop=0.25, ...)
## Default S3 method:
tupleSel(design, type="complete", selprop=0.25, ...)
```

## Arguments

| | |
|---|---|
| design | an experimental design. This can either be a matrix or a data frame in which all columns are experimental factors, or a special data frame of class design, which may also include response data.<br>In any case, the design should be a factorial design; the functions are not useful for quantitative designs (like e.g. latin hypercube samples). |
| with.blocks | a logical, indicating whether or not an existing block factor is to be included into word counting. This option is ignored if design is not of class design.<br>Per default, an existing block factor is ignored.<br>For designs without a block factor, the option does not have an effect.<br>If the design is blocked, and with.blocks is TRUE, the block factor is treated like any other factor in terms of word counting. |
| J | a logical, indicating whether or not a vector of contributions from individual degrees of freedom is produced. If TRUE, the entries of the vector are absolute normalized J-characteristics from all 3- or 4-factor products respectively, based on normalized Helmert contrasts (cf. Ai and Zhang 2004).<br>This is not expected to be useful for practical purposes.<br>J cannot be TRUE simultaneously with separate or rela. |
| rela | logical indicating whether the word lengths are to be calculated in absolute terms (as usual) or relative to the maximum possible word length in case of complete aliasing; if TRUE, each word length is divided by the worst case word length (that |

corresponds to a completely aliased set of factors) derived in Groemping (2011). `rela=TRUE` is only permitted for the shortest possible word length, i.e. `length3` or `P3.3` for resolution III designs, `length4` or `P4.4` for resolution IV designs, or `length5` for resolution V designs.
`rela` cannot be `TRUE` simultaneously with `parft`, `parftdf`, `J` or `separate`.

`separate` a logical, indicating whether or not separate (and overlapping) sums are requested for each two-factor interaction; the idea is to be able to identify clear two-factor interactions; this may be useful for a design for which `length3` returns zero, in analogy to clear 2fis for regular fractional factorials, implemented in function `FrF2`; this is currently experimental and may be removed again if it does not prove useful.
`separate` cannot be `TRUE` simultaneously with `J`.

`n` integer;
for functions `contr.XuWu` and `contr.XuWuPoly`: number of levels of the factor for which to determine contrasts
for function `nchoosek`: number of elements to choose from

`contrasts` must always be `TRUE`; option needed for function `model.matrix` to work properly

`ID` an orthogonal array, either a matrix or a data frame; need not be of class `oa`; can also be a character string containing the name of an array listed in data frame `oacat`

`nlevels` a vector of requested level informations (vector with an entry for each factor)

`all` logical; if `FALSE`, the search stops whenever a design with 0 generalized words of highest requested length is found; otherwise, the function always determines all best designs

`variants` matrix of integer column number entries; each row gives the column numbers for one variant to be compared; the matrix columns must correspond to the entries of the `nlevels` option

`crit` character string that requests `"total"` or `"worst"` triple optimization; `"total"` corresponds to the previous version that optimizes the overall number of length 3 words; `"worst"` minimizes the aliasing of the worst triple.

`min3` the outcome of a call to `oa.min3` with `crit="total"`, which is to be used for a call to `oa.min34`

`maxGR` the outcome of a call to `oa.min3` with `crit="worst"` and `rela=TRUE` (or the outcome of a call to `oa.maxGR`), which is to be used for a call to `oa.minRelProjAberr`

`digits` number of decimal points to which to round the result

`parft` logical indicating whether to tabulate projection averaged $R^2$ values (see Groemping 2013) instead of word lengths; if `TRUE`, the table shows projection averaged $R^2$ values as detailed in Groemping (2013, 2017).
`parft=TRUE` is only permitted for the shortest possible word length, i.e. `length3` or `P3.3` for resolution III designs, `length4` or `P4.4` for resolution IV designs.
`parft` cannot be `TRUE` simultaneously with `rela` or `parftdf`.

`parftdf` logical indicating whether to tabulate averaged $R^2$ values, where averaging is over individual degrees of freedom; this variant is not explicitly described in

Groemping (2013, 2017) and usually yields very similar results as parft, except for some situations where there are factors with very unequal numbers of levels (e.g. 2-level and 8-level factors).
parftdf=TRUE is only permitted for the shortest possible word length, i.e. length3 or P3.3 for resolution III designs, length4 or P4.4 for resolution IV designs.
parftdf cannot be TRUE simultaneously with rela or parft.

detailed — logical indicating whether the vector of all (relative) tuple word lengths is to become an attribute of the output object (attribute detail);
intended for supporting other functions (can be a very long vector!)

k — number of elements to be chosen, integer from 0 to n

type — character string with type of worst case to consider;
"complete", "worst" and "worst.rel" are available.
For a resolution R design, tuples of R factors are considered (works for R=3 and R=4 only). "complete" selects all tuples with complete aliasing of at least one factor, "worst" selects all tuples whose number of words is larger than the 1-selprop quantile of the word length distribution of R-tuples, "worst.rel" does the same with relative words (i.e. increases the weight for tuples whose minimum number of levels is small), "worst.parft" and "worst.parftdf" do the same with the different versions of projection average $R^2$ values.

selprop — (approximate) proportion of worst case tuples to be selected (see type)

... — further arguments; for the design method of function lengths, the defaults with.blocks = FALSE, J = FALSE can be changed here; for function tupleSel, ... is currently not used

## Details

These functions work for factors only and are not intended for quantitative variables. Nevertheless it is possible to apply them to class design plans with quantitative variables in them in some situations.

The generalized word length pattern as introduced in Xu and Wu (2001) is the basis for the functions described here. Consult their article or Groemping (2011) for rigorous mathematical detail of this concept. A brief explanation is also given here, before explaining the details for the functions:
Assume a design with qualitative factors, for which all factors are coded with specially normalized Helmert contrasts (which orthogonalizes the model matrix columns to the intercept column). Functions contr.XuWu and contr.XuWuPoly provide such contrasts based on Helmert contrasts or orthogonal polynomial contrasts, normalized according to the prescription by Xu and Wu (2001) which implies that all model matrix columns have Euclidean norm sqrt(n), provided that each individual factor is balanced.
Then, the number of generalized words of length 3 is determined by taking the sum of squares of the column averages of all three-factor interaction columns (from a model matrix with all three-factor interactions included).
Likewise, the number of generalized words of length 4 is determined by taking the sum of squares of the column averages of all four-factor interaction columns (from a model matrix with all four-factor interactions included), and so on.
A certain plausibility can be found in these numbers by noting that they provide the more well-known word length pattern for regular fractional factorial 2-level designs, implying that they are

exactly zero for resolution IV or resolution V fractional factorial 2-level designs, respectively. Furthermore, Groemping and Xu (2014) provided an interpretation in terms of $R^2$-values from linear models for the number of shortest words.

Function `lengths` calculates the generalized word length pattern (numbers of generalized words of lengths 2, 3, 4 and 5 respectively), functions `length2`, `length3`, `length4` and `length5` calculate each length separately. For designs with few rows and many columns, the newer function `GWLP` is much faster; therefore it will be a better choice than `lengths` for most applications. On the other hand, for designs with many rows, `lengths` can be much faster. Furthermore, `lengths` and the compoment functions `length2` to `length5` can calculate additional detail not available from `GWLP`.

The most important component length functions are `length3` and `length4`; `length2` should yield zero for all orthogonal arrays, and `length5` will in most cases not be of interest either. The number of shortest possible words, e.g. length 4 for resolution IV designs, can be calculated in relative terms, if interest is in the extent of complete aliasing (cf. Groemping 2011).
The length functions are fast for small numbers of factors but can take a long time if the number of factors is large. Note that an orthogonal array based design is called resolution III if the result of function `length3` is non-zero, resolution IV, if the result of function `length3` is zero and the result of function `length4` is non-zero, and resolution V+ (at least V), if the result of both functions `length3` and `length4` are zero.

Functions `P3.3` and `P4.4` calculate the pattern of generalized words of length 3 for all three-factor projections of an array and of generalized words of length 3 or 4 for all four-factor projections of an array. Calculation of such projection frequency tables has been proposed by Xu, Cheng and Wu (2004). The relative version for `P3.3` and `P4.4` has been introduced by Groemping (2011) for better assessment of the projective properties of a design. It divides each absolute number of words by the maximum possible number in case one factor is completely determined by the combinations of the other two factors. For `P4.4`, the relative version is valid only for resolution IV designs. NOTE: For mixed-level designs, it is meanwhile recommended to use ARFTs (Groemping 2013, 2017) instead of relative `P3.3` and `P4.4`; these can be obtained by functions `GRind` or `SCFTs` and have relevant advantages over the projection frequency tables from `P3.3` and `P4.4` for mixed level designs. SCFTs (also treated in Groemping 2013, 2017) provide more detail than ARFTs and are interesting for assessing the suitability of a design for screening purposes.

The functions can be used in selecting among different possibilities to accomodate factors within a given orthogonal array (cf. examples). For general purposes, it is recommended to use designs with as small an outcome of `length3` as possible (either absolute or relative, either total or worst case), and within the same result for `length3` (particularly 0), with as small a result for `length4` as possible. This corresponds to (a step towards) generalized minimum aberration. It can also be useful to consider the patterns, particularly `P3.3`, or for mixed levels the aforementioned ARFTs or SCFTs obtainable with functions `GRind` or `SCFTs`. Note that some overall information on a design's behavior is available in the catalogue data frames `oacat` and `oacat3` and can be queried with function `show.oas`; this helps for selecting a suitable array from which to start optimization efforts (see below).

Functions `oa.min3`, `oa.min4`, `oa.min34` optimize column allocation for a given array for which a certain factor combination must be accomodated: They return designs that allocate columns such that the number of generalized words of length 3 is minimized (`oa.min3`; with a choice between minimizing the total number or minimizing the number for the worst-case triple of factors), or the number of generalized words of length 4 is minimized within all designs for which the number of generalized words of length 3 is minimal (`oa.min34`, total number only); `oa.min4` does the same as `oa.min3`, but for designs of resolution IV, either entirely (e.g. designs from `oacat3`) or through the

selection of suitable column variants. Option rela allows to switch from the default consideration of absolute numbers of words to relative numbers of words according to Groemping (2011). This relative number corresponds to concentrating on the worst-case ARFT entry for each set of R factors (R the resolution).

Function oa.maxGR maximizes generalized resolution according to Deng and Tang (1999) as generalized by Groemping (2011). **Note that function oa.maxGR can be replaced by the much faster function oa.min3 with options crit="worst" and rela=TRUE, whenever GR<=4. Only for designs with GR > 4, the extra effort with function oa.maxGR is useful.**

Function oa.minRelProjAberr conducts minimum relative projection aberration according to Groemping (2011), with the four steps
(a) maximize GR (using function oa.min3 with options crit="worst" and rela=TRUE),
(b) minimize rA3 or rA4 (depending on resolution),
(c) optimize RPFT (as obtained by P3.3 or P4.4) and
(d) minimize absolute words of lengths 4 etc. (only carried through to length 4 by the function).

Functions oa.max3 and oa.max4 do the opposite: they search for the worst design in terms of the number of generalized words of lengths 3 or 4. Such a design can e.g. be used for demonstrating the benefit of optimizing the number of words, or for exemplifying theoretical properties. Occasionally, it may also be useful, if there are severe restrictions on possible combinations. (oa.max4 should only be used for resolution IV designs.)

Function tupleSel selects worst case tuples of R factors for resolution R designs. Depending on the type requested, all completely aliased tuples are selected, or the worst case tuples that exceed the 1-selprop quantile of the numbers of absolute or relative words are selected.

## Value

The functions length3 and length4 (currently) per default return the number of generalized words. If option J=TRUE is set, their value is a named vector of normalized absolute J-characteristics (cf. Ai and Zhang 2004) for the respective length, based on normalized Helmert contrasts, with names indicating factor indices. (For blocked designs with the with.blocks=TRUE option, the block factor has index 1.)

Functions P3.3 and P4.4 return a matrix with the numbers of generalized words of length 3 (4) that do occur for 3 (4) factor projections (column length3 or length4 resp.) and their frequencies. If option rela=TRUE is set, the numbers of generalized words are normalized by dividing them by the number of words that corresponds to perfect aliasing among the factors for each projection. For P4.4, the relative version is only reasonable for resolution IV designs. The matrix of projection frequencies has the overall number of generalized words of the respective length as an attribute; in the case rela=TRUE it also has the generalized resolution and the overall absolute number of generalized words of the respective length as an attribute.

The functions oa.min3, oa.min34, oa.max3 and oa.max4 (currently) return a list with elements GWP (the number(s) of generalized words of length 3 (lengths 3 and 4)) column.variants (the columns to be used for design creation, ordered with ascending nlevels) and complete (logical indicating whether or not the list is guaranteed to be complete). oa.min3, the name of the first element is either GWP3 (crit="total"), worst.a3 (rela=FALSE, crit="worst") or GR (rela=FALSE, crit="worst"). The function oa.maxGR returns a list with elements GR, column.variants and complete, the function oa.minRelProjAberr returns a list with elements GR, GWP, column.variants and complete.

The function nchoosek returns a matrix with k rows and choose(n, k) columns, each of which contains a different subset of k elements.

The function `tupleSel` returns a sorted list of worst case tuples, beginning with the worst case. In case of types `"worst"` or `"worst.rel"`, attributes provide the (relative) projection frequency tables and the sorted vector of the worst case projection values corresponding to the listed tuples.

## Warning

The functions have been checked on the types of designs for which they are intended (especially orthogonal arrays produced with oa.design) and on 2-level fractional factorial designs produced with package **FrF2**. They may produce meaningless results for some other types of designs.

Furthermore, all optimizing functions work for relatively small problems only and will break down for larger problems because of storage space requirements (size depends on the number of possible selections among columns; for example, selecting 9 out of 31 columns is not doable on my computer because of storage space issues, while selecting 29 out of 31 columns is doable within the available storage space). Programming of a less storage-intensive algorithm is underway.

## Note

Function `nchoosek` has been taken from **Bioconductor** package **vsn**.
Function `GWLP` is much faster (but also more inaccurate) than function `lengths` and may be a better choice for designs with many factors.

## Author(s)

Ulrike Groemping

## References

Ai, M.-Y. and Zhang, R.-C. (2004). Projection justification of generalized minimum aberration for asymmetrical fractional factorial designs. *Metrika* **60**, 279–285.

Groemping, U. (2011). Relative projection frequency tables for orthogonal arrays. Report 1/2011, *Reports in Mathematics, Physics and Chemistry* http://www1.bht-berlin.de/FB_II/reports/welcome.htm, Department II, Berliner Hochschule fuer Technik (formerly Beuth University of Applied Sciences), Berlin.

Groemping, U. (2013). Frequency tables for the coding invariant ranking of orthogonal arrays. Report 2/2013, *Reports in Mathematics, Physics and Chemistry* http://www1.bht-berlin.de/FB_II/reports/welcome.htm, Department II, Berliner Hochschule fuer Technik (formerly Beuth University of Applied Sciences), Berlin.

Groemping, U. (2017). Frequency tables for the coding invariant quality assessment of factorial designs. *IISE Transactions* **49**(5), 505-517. doi: 10.1080/0740817X.2016.1241458.

Xu, H.-Q. and Wu, C.F.J. (2001). Generalized minimum aberration for asymmetrical fractional factorial designs. *The Annals of Statistics* **29**, 1066–1077.

Xu, H., Cheng, S., and Wu, C.F.J. (2004). Optimal projective three-level designs for factor screening and interaction detection. *Technometrics* **46**, 280–292.

## See Also

See also GWLP for a version of lengths that is much faster for designs with not so many runs, and GRind for another set of quality criteria for orthogonal arrays.
Package **DoE.MIParray** can create arrays for smallish situations for which the catalogued arrays do not provide satisfactory results; this package requires at least one of the commercial softwares Mosek or Gurobi to be installed (both provide free academic licenses).

## Examples

```
## check a small design
oa12 <- oa.design(nlevels=c(2,2,6))
length3(oa12)
## length4 is of course 0, because there are only 3 factors
P3.3(oa12)

## the results need not be an integer
oa12 <- oa.design(L12.2.11,columns=1:6)
length3(oa12)
length4(oa12)
P3.3(oa12)  ## all projections have the same pattern
          ## which is known to be true for the complete L12.2.11 as well
P3.3(L18)   ## this is the pattern of the Taguchi L18
          ## also published by Schoen 2009
P3.3(L18[,-2])  ## without the 2nd column (= the 1st 3-level column)
P3.3(L18[,-2], rela=TRUE)  ## relative pattern, divided by theoretical upper
                           ## bound for each 3-factor projection

## choosing among different assignment possibilities
## for two 2-level factors and one 3- and 4-level factor each
show.oas(nlevels=c(2,2,3,4))
## default allocation: first two columns for the 2-level factors
oa24.bad <- oa.design(L24.2.13.3.1.4.1, columns=c(1,2,14,15))
length3(oa24.bad)
## much better: columns 3 and 10
oa24.good <- oa.design(L24.2.13.3.1.4.1, columns=c(3,10,14,15))
length3(oa24.good)
length4(oa24.good)  ## there are several variants,
                    ## which produce the same pattern for lengths 3 and 4

## the difference matters
plot(oa24.bad, select=c(2,3,4))
plot(oa24.good, select=c(2,3,4))

## generalized resolution differs as well (resolution is III in both cases)
GR(oa24.bad)
GR(oa24.good)

## and analogously also GRind and ARFT and SCFT
GRind(oa24.bad)
GRind(oa24.good)

## GR and GRind can be different
```

```
GRind(L18[, c(1:4,6:8)], arft=FALSE, scft=FALSE)

## choices for columns can be explored with functions oa.min3, oa.min34 or oa.max3
oa.min3(L24.2.13.3.1.4.1, nlevels=c(2,2,3,4))
oa.min34(L24.2.13.3.1.4.1, nlevels=c(2,2,3,4))
## columns for designs with maximum generalized resolution
##    (can take very long, if all designs have worst-case aliasing)
   ## then optimize these for overall relative number of words of length 3
   ##     and in addition absolute number of words of length 4
mGR <- oa.maxGR(L18, c(2,3,3,3,3,3,3))
oa.minRelProjAberr(L18, c(2,3,3,3,3,3,3), maxGR=mGR)

oa.max3(L24.2.13.3.1.4.1, nlevels=c(2,2,3,4))    ## this is not for finding
                                                 ## a good design!!!

## Not run:
## play with selection of optimum design
## somewhat experimental at present
oa.min3(L32.2.10.4.7, nlevels=c(2,2,2,4,4,4,4,4))
best3 <- oa.min3(L32.2.10.4.7, nlevels=c(2,2,2,4,4,4,4,4), rela=TRUE)
oa.min34(L32.2.10.4.7, nlevels=c(2,2,2,4,4,4,4,4))
oa.min34(L32.2.10.4.7, nlevels=c(2,2,2,4,4,4,4,4), min3=best3)

## generalized resolution according to Groemping 2011, manually
best3GR <- oa.min3(L36.2.11.3.12, c(rep(2,3),rep(3,3)), rela=TRUE, crit="worst")
   ## optimum GR is 3.59
## subsequent optimization w.r.t. rA3
best3reltot.GR <- oa.min3(L36.2.11.3.12, c(rep(2,3),rep(3,3)), rela=TRUE,
        variants=best3GR$column.variants)
   ## optimum rA3 is 0.5069
## (note: different from first optimizing rA3 (0.3611) and then GR (3.5))
## remaining nine designs: optimize RPFTs
L36 <- oa.design(L36.2.11.3.12, randomize=FALSE)
lapply(1:9, function(obj) P3.3(L36[,best3reltot.GR$column.variants[obj,]]))
   ## all identical
oa.min34(L36, nlevels=c(rep(2,3),rep(3,3)), min3=best3reltot.GR)
   ## still all identical

## End(Not run)

   ## select among column variants with projection frequencies
   ## here, all variants have identical projection frequencies
   ## for larger problems, this may sometimes be relevant
   variants <- oa.min34(L24.2.13.3.1.4.1, nlevels=c(2,2,3,4))
   for (i in 1:nrow(variants$column.variants)){
      cat("variant ", i, "\n")
      print(P3.3(oa.design(L24.2.13.3.1.4.1, columns=variants$column.variants[i,])))
      }

   ## automatic optimization is possible, but can be time-consuming
   ## (cf. help for oa.design)
   plan <- oa.design(L24.2.13.3.1.4.1, nlevels=c(2,2,3,4), columns="min3")
   length3(plan)
```

```
length4(plan)
plan <- oa.design(L24.2.13.3.1.4.1, nlevels=c(2,2,3,4), columns="min34")
length3(plan)
length4(plan)

## Not run:
## blocked design from FrF2
## the design is of resolution IV
## there is one (generalized) 4-letter word that does not involve the block factor
## there are four more 4-letter words involving the block factor
## all this and more can also be learnt from design.info(plan)
require(FrF2)
plan <- FrF2(32,6,blocks=4)
length3(plan)
length3(plan, with.blocks=TRUE)
length4(plan)
length4(plan, with.blocks=TRUE)
design.info(plan)

## End(Not run)
```

---

`getblock` *Functions to extract a block factor from a class design object or to rerandomize a class design object*

---

## Description

Function getblock creates block factors for designs with replications, repeated measurements or split plot designs. Function rerandomize.design rerandomizes an experimental design.

## Usage

```
getblock(design, combine=FALSE, ...)
rerandomize.design(design, seed=NULL, block=NULL, ...)
```

## Arguments

| | |
|---|---|
| `design` | an object of class `design`, which is a design with replications or repeated measurements or a split plot design |
| `combine` | logical with default `FALSE`. It has an effect for replicated blocked and splitplot designs only: If `TRUE`, all blocking information is combined into a single factor. Otherwise, a data frame with separate identifiers is returned. |
| `seed` | integer number for initialization of the random number generator (needed for repeatable rerandomization)<br>In R version 3.6.0 and later, the default behavior of function `sample` has changed. If you work in a new (i.e., >= 3.6.-0) R version and want to reproduce a rerandomization from an earlier R version (before 3.6.0), you have to change the |

RNGkind setting by
`RNGkind(sample.kind="Rounding")`
before running function `rerandomize.design`.
It is recommended to change the setting back to the new recommended way afterwards:
`RNGkind(sample.kind="default")`
For an example, see the documentation of the example data set `VSGFS`.

`block` character string giving the name of a block factor (only for unreplicated designs that do not have any prior blocking or split plot structure; meant for block randomization of designs created with function `oa.design`)

`...` currently not used

## Details

The purpose of function `getblock` is to support users in doing their own analyses accomodating randomization restrictions like blocking and split plotting with R modeling functions.

The reason for including designs with proper replications is that these are randomized in blocks by packages **DoE.base** and **FrF2** and partly by **DoE.wrapper**. While the package author does not consider it generally necessary to analyze these with a block factor, function getblock makes it easy for users with a different opinion (or for situations for which time turns out to be important in spite of not having explicitly blocked for time) to run an analysis with a block factor for the replication.

For unreplicated split plot designs, a whole plot identifier is returned; the design itself contains the plot information via the settings of the whole plot factors only. Thus, it may be useful to be able to create the plot identifier.

For replicated block or split plot designs, there is a randomization hierarchy that will depend on how the experiment was actually conducted. Therefore, a dataframe is generated the columns of which can be used in the appropriate way by a statistically literate user.

Function `rerandomize.design` rerandomizes a design. This can be useful if the user wants to obtain unblocked replications (packages `DoE.base` and **FrF2** usually randomize in blocks on time) or wants to freely randomize the center point position over the whole range of the experiment (or a block, respectively), or if the user wants to also randomize the blocks (rather than randomizing the block units to the experimental blocks outside of the design), or if the user wants to do block randomization on a block factor specified with the `block` option for a design created with function `oa.design` or `pb` (which do not offer explicit specification of blocking).
It can also be useful for ensuring a randomization that has little correlation between run order and model matrix columns; this correlation can e.g. be checked with the help of function `corrPlot`, using the option `run.order=TRUE`.

## Value

Function `getblock` returns
a single factor with block information (for split plot designs without replication or replicated designs without randomization restrictions)
or a data frame with several blocking factors (for designs with randomization restrictions and replication).

Function rerandomize.design returns a class design object; note that it will not be possible to add center points after re-randomization, i.e. if required, center points have to be added before using the function.

## Warning

Since R version 3.6.0, the behavior of function sample has changed (correction of a biased previous behavior that should not be relevant for the randomization of designs). For reproducing a re-randomization that was produced with an earlier R version, please follow the steps described with the argument seed.

## Author(s)

Ulrike Groemping

## Examples

```
 ## a blocked full factorial design
ff <- fac.design(nlevels=c(2,2,2,3,3,3), blocks=6, bbrep=2, wbrep=2, repeat.only=FALSE)
 getblock(ff)
 getblock(ff, combine=TRUE)
 rerandomize.design(ff)
 ff <- fac.design(nlevels=c(2,2,2,3,3,3), replications=2, repeat.only=FALSE)
 getblock(ff)
 ff <- fac.design(nlevels=c(2,2,2,3,3,3), replications=2, repeat.only=FALSE)
 try(getblock(ff))
 ## a design created with oa.design
 small <- oa.design(nlevels=c(2,2,2,2,2,2,2,2,8))
 rerandomize.design(small, block="J")
```

---

GRind — *Functions for calculating generalized resolution, average R-squared values and squared canonical correlations, and for checking design regularity*

---

## Description

Function GR calculates generalized resolution, function GRind calculates more detailed generalized resolution values, squared canonical correlations and average R-squared values, the print method for class GRind appropriately prints the detailed GRind values. Function SCFTs calculates squared canonical correlations for factorial designs. SCFTs includes more projections than GRind (all full resolution projections or even all projections) and decides on regularity of the design, based on a conjecture.

## Usage

```
GR(ID, digits=2)
GRind(design, digits=3, arft=TRUE, scft=TRUE, cancors=FALSE, with.blocks=FALSE)
SCFTs(design, digits = 3, all = TRUE, resk.only = TRUE, kmin = NULL, kmax = ncol(design),
   regcheck = FALSE, arft = TRUE, cancors = FALSE, with.blocks = FALSE)
## S3 method for class 'GRind'
print(x, quote=FALSE,  ...)
```

## Arguments

| | |
|---|---|
| ID | an orthogonal array, either a matrix or a data frame; need not be of class oa; can also be a character string containing the name of an array listed in data frame oacat |
| digits | number of decimal points to which to round the result |
| design | a factorial design. This can either be a matrix or a data frame in which all columns are experimental factors, or a special data frame of class design, which may also include response data.<br>In any case, the design should be a factorial design; the functions are not useful for quantitative designs (like e.g. latin hypercube samples). |
| arft | logical indicating whether or not the average $R^2$ frequency table (ARFT, see Groemping 2013) is to be returned |
| scft | logical indicating whether the squared canonical correlation frequency table (SCFT, see Groemping 2013) is to be returned |
| cancors | logical indicating whether individual canonical correlations are to be returned (see Groemping 2013). These will not be needed for normal use of the package. |
| with.blocks | a logical, indicating whether or not an existing block factor is to be included into word counting. This option is ignored if design is not of class design.<br>Per default, an existing block factor is ignored.<br>For designs without a block factor, the option does not have an effect.<br>If the design is blocked, and with.blocks is TRUE, the block factor is treated like any other factor in terms of word counting. |
| all | logical; decides whether or not to consider projections of more than R~factors, where R denotes the design resolution |
| resk.only | logical; if all is TRUE, should only full resolution projections be considered? Choosing FALSE may cause very long run times. |
| kmin | integer; purpose is to continue an earlier run with additional larger projections |
| kmax | integer; limit on projection sizes to consider |
| regcheck | logical; is the purpose a regularity check? If TRUE, the function stops after the first projection size that included squared canonical correlation different from 0 or 1. |
| x | a list of class GRind, as created by function GRind |
| quote | a logical indicating whether character values are quoted |
| ... | further arguments to function print |

## Details

Functions GR, GRind, and SCFTs work for factors only and are not intended for quantitative variables. Nevertheless it is possible to apply them to class design plans with quantitative variables in them in some situations.

Function GR calculates the generalized resolution according to Deng and Tang (1999) for 2-level designs or a generalization thereof according to Groemping (2011) and Groemping and Xu (2014) for general orthogonal arrays. It returns a value between 3 and 5, where the numeric value 5 stands for "at least 5". Roughly, generalized resolution measures the closeness of a design to the next higher resolution (worst-case based, e.g. one completely aliased triple of factors implies resolution 3).

Function GRind (newer than GR, and recommended) calculates the generalized resolution, together with factor wise generalized resolution values, squared canonical correlations and average R-squared values, as mentioned in Groemping and Xu (2014) and further developed in Groemping (2013, 2017). The print method for class Grind objects prints the individual factor components of GRind.i such that they do not mislead: Because of the shortest word approach for GR, SCFT and ARFT, a GRind.i component can be at most one larger than the resolution. For example, if GR is 3.5 so that the resolution is 3, the largest possible numeric value of a GRind.i component is 4, but it means ">=4".

Function SCFTs does more extensive SCFT and ARFT calculations than function GRind: in particular, the function allows to do such calculations for more projection sizes, either restricting attention to full resolution projections or going for ALL projections with non-zero word lengths. These capabilities have been introduced in relation to regularity checking based on SCFTs (see Groemping and Bailey 2016): Defining a factorial design as regular if all main effects are orthogonal in some sense to effects including other factors of any order, it is conjectured that a regularity check on full resolution projections only will suffice for identifying non-regularity (work in progress). However, this is a conjecture only; as long as it is not proven, a definite check for this type of regularity requires checking ALL projections, i.e. setting resk.only to FALSE. With this setting, the function may run for a very long time (depends in particular on the number of factors)!

## Value

Function GR returns a list with elements GR (the generalized resolution of the array, a not necessarily integer number between 3 and 5) and RPFT (the relative projection frequency table). GR values smaller than 5 are exact, while the number five stands for "at least 5". The resolution itself is the integer portion of GR. The RPFT element is the relative projection frequency table for 4-factor projections for GR=5. For unconfounded three- and four-column designs, GR takes the value Inf (used to be 5 for package versions up to 0.23-4).

Function GRind works on designs with resolution at least 3 and returns a list with elements
GRs (the two versions of generalized resolution described in Groemping and Xu 2014),
the matrix GR.i with rows GRtot.i and GRind.i for the factor wise generalized resolutions (also in Groemping and Xu 2014),
and optionally
the ARFT (Groemping 2013, 2017),
the SCFT (Groemping 2013, 2017),
and/or the canonical correlations.
The latter are held in an

nfac x choose(nfac-1, R-1) x max(nlev)-1 array
and are supplemented with 0es, if there are fewer of them than the respective dfi.

The factor wise generalized resolutions are in the closed interval between resolution and resolution + 1. In the latter case, their meaning is "at least resolution + 1". (The print method ensures that they are printed accordingly, but the list elements themselves are just the numbers.)

Function `SCFTs` returns a list of lists with a component for each projection size considered. Each such component contains the following entries:

| | |
|---|---|
| `SCFT` | Squared canonical correlation table for the projection size |
| `ARFT` | Average $R^2$ frequency table for the projection size (if requested) |
| `cancors` | canonical correlations (if requested) |

## Warning

The functions have been checked on the types of designs for which they are intended (especially orthogonal arrays produced with oa.design) and on 2-level fractional factorial designs produced with package **FrF2**. They may produce meaningless results for some other types of designs.

## Author(s)

Ulrike Groemping

## References

Groemping, U. (2011). Relative projection frequency tables for orthogonal arrays. Report 1/2011, *Reports in Mathematics, Physics and Chemistry* http://www1.bht-berlin.de/FB_II/reports/welcome.htm, Department II, Berliner Hochschule fuer Technik (formerly Beuth University of Applied Sciences), Berlin.

Groemping, U. (2013). Frequency tables for the coding invariant ranking of orthogonal arrays. Report 2/2013, *Reports in Mathematics, Physics and Chemistry* http://www1.bht-berlin.de/FB_II/reports/welcome.htm, Department II, Berliner Hochschule fuer Technik (formerly Beuth University of Applied Sciences), Berlin.

Groemping, U. (2017). Frequency tables for the coding invariant quality assessment of factorial designs. *IISE Transactions* **49**, 505-517. doi: 10.1080/0740817X.2016.1241458.

Groemping, U. and Bailey, R.A. (2016). Regular fractions of factorial arrays. In: *mODa 11 – Advances in Model-Oriented Design and Analysis*. New York: Springer.

Groemping, U. and Xu, H. (2014). Generalized resolution for orthogonal arrays. *The Annals of Statistics* **42**, 918–939. https://projecteuclid.org/euclid.aos/1400592647

## See Also

See also `GWLP` and `generalized.word.length`

## Examples

```
oa24.bad <- oa.design(L24.2.13.3.1.4.1, columns=c(1,2,14,15))
oa24.good <- oa.design(L24.2.13.3.1.4.1, columns=c(3,10,14,15))
## generalized resolution differs (resolution is III in both cases)
GR(oa24.bad)
GR(oa24.good)

## and analogously also GRind and ARFT and SCFT
GRind(oa24.bad)
GRind(oa24.good)

## SCFTs
## Not run: plan <- L24.2.12.12.1[,c(1:5,13)]
GRind(plan)  ## looks regular (0/1 SCFT only)
SCFTs(plan)
SCFTs(plan, resk.only=FALSE)

## End(Not run)
```

---

GWLP *Function for fast calculation of GWLP*

---

## Description

Calculates GWLP using the formulae from Xu and Wu (2001)

## Usage

```
GWLP(design, ...)
## S3 method for class 'design'
GWLP(design, kmax=design.info(design)$nfactors,
     attrib.out=FALSE, with.blocks = FALSE, digits = NULL, ...)
## Default S3 method:
GWLP(design, kmax=ncol(design), attrib.out=FALSE, digits = NULL, ...)
```

## Arguments

| | |
|---|---|
| design | a design, not necessarily of class design; class design properties are exploited by using only factor columns (or factor and block columns, if with.blocks is TRUE) |
| kmax | the maximum word length requested |
| attrib.out | the detail added to the output (see Value section) |
| with.blocks | if TRUE, the block column contributes to the GWLP, otherwise it does not |
| digits | the number of decimals to round to; NULL prevents rounding |
| ... | further arguments to generic GWLP; not used in the methods |

## Details

Function GWLP is much faster but also more inaccurate than the function lengths, which calculates numbers of words for lengths 2 to 5 only. Note, however, that function lengths can be faster for designs with very many rows.
If a design factor contains only some of the intended levels, design must be a data frame, and the factor must be an R factor with the complete set of levels specified, in order to make function GWLP aware of the missing levels.

## Value

The GWLP methods output a named vector with the numbers of generalized words of lengths zero to kmax. If attrib.out is TRUE, this vector comes with the attributes B and levels.info, the latter documenting the level situation of the design, the former the distance distribution B (Xu and Wu 2001).

## Author(s)

Hongquan Xu, Ulrike Groemping

## References

Xu, H.-Q. and Wu, C.F.J. (2001). Generalized minimum aberration for asymmetrical fractional factorial designs. *Annals of Statistics* **29**, 1066–1077.

## See Also

See Also lengths

## Examples

```
GWLP(L18)
GWLP(L18, attrib.out=TRUE)
```

---

# halfnormal *Creation of half normal effects plots and numeric methods for significance assessment*

---

## Description

Generic function and methods for creating half normal effects plots

## Usage

```
halfnormal(x, ...)
## Default S3 method:
halfnormal(x, labs=names(x), codes = NULL, pch = 1, cex.text = 1,
   alpha = 0.05, xlab = "absolute effects", large.omit = 0, plot=TRUE,
   crit=NULL, ...)
## S3 method for class 'lm'
halfnormal(x, labs = NULL, code = FALSE, pch = NULL, cex.text = 1,
   alpha = 0.05, xlab = "absolute coefficients", large.omit = 0, plot=TRUE,
   keep.colons = !code, ME.partial = FALSE,
   external.pe = NULL, external.center = FALSE, contr.center = "contr.poly",
   pch.set = c(1, 16, 8), scl = NULL, method="Lenth",
   legend=code, err.points=TRUE, err.line=TRUE, linecol="darkgray", linelwd=2,
   ...)
## S3 method for class 'design'
halfnormal(x, response = NULL, labs = NULL, code = FALSE, pch = NULL,
   cex.text = 1,
   alpha = 0.05, xlab = "absolute coefficients", large.omit = 0, plot=TRUE,
   keep.colons = !code, ME.partial = FALSE,
   external.pe = NULL, external.center = FALSE, contr.center = "contr.poly",
   pch.set = c(1, 16, 8), scl = NULL, method="Lenth",
   legend=code, err.points=TRUE, err.line=TRUE, linecol="darkgray", linelwd=2,
   ...)

ME.Lenth(b, simulated=TRUE, alpha=NULL)
CME.LW98(b, sterr, dfe, simulated=TRUE, alpha=NULL)
CME.EM08(b, sterr, dfe, simulated=TRUE, weight0=5, alpha=NULL)
```

## Arguments

| | |
|---|---|
| x | a numeric vector of effects, a linear model from experimental data, or an experimental design of class design |
| labs | effect labels;<br>default labels: for the default method, names of the vector x, or b1, b2, ... for unnamed vectors; for classes design or lm taken from the linear model |
| codes | a vector with a code for each effect; the default NULL uses the labs values |
| code | a logical; TRUE implies that factor letters are used instead of factor codes, and that the default for default for keep.colons is changed to FALSE |
| pch | plot symbol; NULL, a number or a vector of plot symbol numbers or the same length as the effects in x;<br>in the default method, a single number (default 1) implies that the given plotting symbol is used for for all points;<br>for the other methods, the default NULL or a single number implies that pch.set is employed for lack of fit or pure error contrast points;<br>for the non-default methods, a vector-valued pch will only rarely be useful (see Details section) |

| | |
|---|---|
| cex.text | factor to hand to cex argument for point labeling with function text and margin annotations with function mtext; for mtext, it is multiplied with par("cex"), in order to obtain the same size for point labels and the margin annotations. |
| alpha | number between 0 and 1: the significance level for labelling effects; for functions ME.Lenth, CME.LW98 and CME.EM08, alpha can also be NULL or a numeric vector; for using the simulated critical bounds, all elements of alpha must be in 0.01,0.02,...,0.25 |
| xlab | character string: the x axis label |
| plot | logical; if FALSE, plotting is suppressed |
| large.omit | integer number of largest effects to be omitted from plot and calculations in order to concentrate on the smaller effects; (note that the significance is also re-assessed; if that is undesirable, an explicit crit value can be specified from all coefficients, or alpha can be adjusted to reflect the same significant effects as with all coefficients) |
| crit | default NULL; not meant for the end user; allows the method option for linear models and experimental designs to choose alternatives to Lenth's method |
| keep.colons | if TRUE, the automatic effect labels contain colons for interactions |
| ME.partial | if TRUE, partial aliasing among main effects is permitted and will be orthogonalized away |
| external.pe | numeric vector with values from outside the experimental data for use in estimating the error variance |
| external.center | if TRUE, external values from external.pe are taken as center point values, and a nonlinearity check contrast is estimated from them |
| contr.center | contrasts used for external center points; contr.poly or contr.XuWu |
| pch.set | plot symbols used for experimental effects, automatically determined lack of fit contrasts or pure error effects |
| scl | squared column length to which the model matrix is normalized; default: number of experimental runs |
| method | the default "Lenth" applies Lenth's method to the combined set of effects including error contrasts (if any); the alternatives "LW98" or "EM08" apply the methods proposed by Larntz and Whitcomb (1998) or Edwards and Mee (2008) with weight0=5; if there is no pure error, method "Lenth" is always used, with a warning |
| legend | squared column length to which the model matrix is normalized; default: number of experimental runs |
| err.points | logical, default TRUE; determines, whether pure error points are added to the plot (lack-of-fit points are always added) |
| err.line | logical, default TRUE; determines, whether null line is added to the plot in case pure error points are available |
| linecol | specifies the color for the null line, if applicable |
| linelwd | specifies the width of the null line, if applicable |
| response | response for which the plot is to be created |

| | |
|---|---|
| ... | further options to be handed to the plot function;<br>among these, if options col and/or cex have an element for each effect, these are used in the expected order (first color refers to first element of x and so forth); this change was introduced in version 0.26-2 and causes an appropriate reordering in the actual plot function. |
| b | vector of coefficients |
| simulated | logical; if FALSE, the original critical values from Lenth 1989 are used, otherwise the methods use stored simulated values from a million simulation runs for significance levels of 0.01, 0.02, ..., to 0.25 |
| sterr | a standard error for b, obtained from (a few, dfe) pure error degrees of freedom; the methods by Larntz and Whitcomb (1998) and Edwards and Mee (2008) combine this with Lenth's method |
| dfe | the number of pure error degrees of freedom on which sterr was based |
| weight0 | a tuning parameter for the method by Edwards and Mee 2008; Edwards and Mee recommend to set this to 5 |

## Details

Function halfnormal creates half normal effects plots with automatic effect labelling according to significance. It also prints the significant effects and creates an output object that contains only the vector if signifcant effects (for the default method) or in addition several further components (see section "Value"). Note: The methods for linear models and experimental designs plot absolute coefficients from a linear model (i.e. in case of 2-level factors with the usual -1/+1 coding, half of the absolute effects).

The methods for linear models and experimental designs allow to automatically create lack of fit and pure error contrasts to also be included in the plot, following an orthogonalization strategy similar to Section 5 in Langsrud (2001). Furthermore, they handle factors with more than two levels, and they handle partially aliased effects by orthogonalizing out previous effects from later effects in the model order (similar to what Langsrud 2001 proposed for multiple response variables); thus, the plots are order dependent in case of partial aliasing. The more severe the partial aliasing, the more drastic the difference between the different effect orders. Per default, main effects are required to be orthogonal; this can be changed via option ME.partial.

The functions ME.Lenth, CME.LW98 and CME.EM08 yield standard error estimates and critical values. For alpha in 0.01, 0.02, ..., 0.25, function ME.Lenth uses simulated critical values from a large number of simulations (1000000), if the number of effects is in 7 to 143. Functions CME.LW98 and CME.EM08 currently simulate critical values from 10000 simulation runs on the fly. If no simulated values are available or simulation has been switched off, the half-normal plotting routines will use the conservative t-values proposed by Lenth (1989) (ME.Lenth) or Larntz and Whitcomb (CME.LW98 and CME.EM08).

Vector valued entries for pch, col and cex are handled very specifically for the class lm and class design methods: They make the most sense if the model is already saturated: If no pure error effects have been automatically calculated, effects whose pch is identical to the third element of pch.set will be treated as pure error effects; this allows to manually code these effects.
Generally, vector-valued pch (and col and cex) must have as many elements as the final coefficients vector after augmenting the coefficients; the coefficient vector carries first the experimental coefficients, then the automatically calculated lack-of-fit coefficients, then the automatically calculated pure error coefficients, then lack-of fit coefficients from external replications, and finally the

pure error coefficients from external replications. Even for `err.points=FALSE`, entries for all these elements are needed. The value for `pch` determines, which coefficients are considered pure error.

## Value

The default method for `halfnormal` visibly returns a character vector of significant effects only. The methods for linear models and experimental designs invisibly return lists of nine elements:

| | |
|---|---|
| `coef` | contains the estimated coefficients |
| `mm` | contains the model matrix<br>after adjustment to equally scaled independent effects |
| `mod.effs` | the effects that are part of the model |
| `res` | list that indicates the effects (named vector of position numbers)<br>that were projected out from any particular model effect (element name) |
| `LCs` | contains the coefficients of the linear combinations<br>taken from the residuals after projecting out the effects<br>listed in `res` from the original model matrix columns.<br>Where `LCs` elements are `NULL`,<br>the original effect completely disappeared<br>because of complete confounding with previous effects. |
| `alpha` | contains the significance level |
| `method` | contains the method of significance assessment |
| `signif` | is a character vector of significant effects |
| `pchs` | is a numeric vector of plot character identifiers |

The functions `ME.Lenth`, `CME.LW98` and `CME.EM08` each return lists of length 4 with an estimate for s0, PSE, ME and SME for Lenth's method or their respective modifications for the other two methods (called s0, CPSE, CME and CSME for `CME.LW98` and Cs0, CPSE, CME and CSME for `CME.EM08`). The length of the (C)ME and (C)SME components depends on the length of alpha (default: 25 critical values for alphas from 0.25 to 0.01).

## Note

If someone worked out how to modify symbol colors (option `col`) and/or sizes (option `cex`) for a version before 0.26-2, version 0.26-2 will mess up the order of the symbol colors and/or sizes. The benefit: colors and symbol sizes can now be specified in the natural order, see description of the ... argument.

## Author(s)

Ulrike Groemping, Berliner Hochschule fuer Technik

## References

Daniel, C. (1959) Use of Half Normal Plots in Interpreting Two Level Experiments. *Technometrics* **1**, 311–340.

Daniel, C. (1976) *Application of Statistics to Industrial Experimentation.* New York: Wiley.

Edwards, D. and Mee, R. (2008) Empirically Determined p-Values for Lenth t Statistics. *Journal of Quality Technology* **40**, 368–380.

Langsrud, O. (2001) Identifying Significant Effects in Fractional Factorial Multiresponse Experiments. *Technometrics* **43**, 415–424.

Larntz, K. and Whitcomb, P. (1998) Use of replication in almost unreplicated factorials. Manuscript of a presentation given at the 42nd ASQ Fall Technical conference in Corning, New York. Downloaded 4/26/2013 at https://cdnm.statease.com/pubs/use-of-rep.pdf.

Lenth, R.V. (1989) Quick and easy analysis of unreplicated factorials. *Technometrics* **31**, 469–473.

## See Also

See also DanielPlot for (half) normal plots of 2-level fractional factorial designs without partial aliasing and ignoring any residual degrees of freedom

## Examples

```
### critical values
b <- rnorm(12)
ME.Lenth(b)
ME.Lenth(b)$ME
ME.Lenth(b, alpha=0.22)
ME.Lenth(b, alpha=0.123)
ME.Lenth(b, alpha=0.12)
ME.Lenth(rnorm(144), alpha=0.1)
(mel <- ME.Lenth(b, alpha=0.1))
## assuming an external effect standard error based on 3df
## Not run: CME.EM08(b, 0.1, 3, alpha=0.1)
         ## does not run for saving CRAN check time
         ## much smaller than Lenth, if external
         ## standard error much smaller than s0 (see mel)

### Half normal plots
## the default method
halfnormal(rnorm(15), labs=paste("b",1:15,sep=""))
b <- c(250, 8,7,6, rnorm(11))
halfnormal(b, labs=paste("b",1:15,sep=""))
halfnormal(b, labs=paste("b",1:15,sep=""), large.omit=1)

## the design method, saturated main effects design
plan <- oa.design(L12.2.11)
halfnormal(add.response(plan,rnorm(12)))

## the design method, saturated main effects design,
## partial aliasing due to a missing value
y <- c(NA, rnorm(11))
## the following line would yield an error, because there is even
## complete aliasing among main effects:
## Not run: halfnormal(lm(y~., add.response(plan, y)), ME.partial=TRUE)
## this can only be helped by omitting a main effect from the model;
## afterwards, there is still partial aliasing,
## which must be explicitly permitted by the ME.partial option:
```

```
halfnormal(lm(y~.-D, add.response(plan, y)), ME.partial=TRUE)

## the linear model method
yc <- rnorm(12)
## partial aliasing only
halfnormal(lm(yc~A+B+C+D+E+F+G+H+J+A:B, plan))
## both partial (A:B) and complete (E:F) aliasing are present
halfnormal(lm(yc~A+B+C+D+E+F+G+H+J+A:B+E:F, plan))
## complete aliasing only because of the missing value in the response
halfnormal(lm(y~A+B+C+D+E+F+G+H+J+A:B+E:F, plan),ME.partial=TRUE)
## omit a large dominating effect

halfnormal(lm(y~A+B+C+D+E+F+G+H+J+A:B+E:F, plan),ME.partial=TRUE)


## a regular fractional factorial design with center points
y20 <- rnorm(20)
## Not run: halfnormal(lm(y20~.^2, FrF2(16,7,ncenter=4)))
```

---

ICFTs *Function for calculating interaction contribution frequency tables*

---

## Description

Function ICFTs calculates interaction contribution frequency tables, function ICFT does the same for an entire (usually small) design with more detail.

## Usage

```
ICFTs(design, digits = 3, resk.only = TRUE, kmin = NULL, kmax = ncol(design),
   detail = FALSE, with.blocks = FALSE, conc = TRUE)
ICFT(design, digits = 3, with.blocks = FALSE, conc = TRUE, recode=TRUE)
```

## Arguments

| | |
|---|---|
| design | a factorial design. This can either be a matrix or a data frame in which all columns are experimental factors, or a special data frame of class design, which may also include response data.<br>In any case, the design should be a factorial design; the functions are not useful for quantitative designs (like e.g. latin hypercube samples). |
| digits | integer; number of digits to round to |
| resk.only | logical; if all is TRUE, should only full resolution projections be considered? Choosing FALSE may cause very long run times. |
| kmin | integer; purpose is to continue an earlier run with additional larger projections |
| kmax | integer; limit on projection sizes to consider |
| detail | logical indicating whether calculation details are to be returned (see Groemping 2016). These will not be needed for normal use of the outcome, but may be interesting for special situations. |

| | |
|---|---|
| with.blocks | a logical, indicating whether or not an existing block factor is to be included into word counting. This option is ignored if design is not of class design.<br>Per default, an existing block factor is ignored.<br>For designs without a block factor, the option does not have an effect.<br>If the design is blocked, and with.blocks is TRUE, the block factor is treated like any other factor. |
| conc | logical indicating whether ambiguities should be resolved concentrating the contribution on as few individual values as possible (default) or distributing it as evenly as possible (if FALSE) |
| recode | logical indicating whether or not to recode each column into normalized orthogonal coding with function contr.XuWu;<br>if set to FALSE, it is the users responsibility to provide a design in a *normalized orthogonal* coding of choice |

## Details

The functions work for factors only and are not intended for quantitative variables.

Function ICFTs decomposes the projected $a_k$ values (most often: projected $a_3$ values) into single degree of freedom contributions from the respective $k$ factor interaction.

Function ICFT decomposes the all-factor interaction of the design given to it; it is intended for deep-dive investigations.

The ICFT itself is independent of the choice of normalized orthogonal coding, as are the singular values and the matrix of left singular vectors; in case of several identical singular values, the left singular vectors are not uniquely determined but are subject to arbitrary rotation. The right singular vectors depend on the choice of normalized orthogonal coding. They represent the directions of coefficient vectors for which the interaction contributions indicate the bias potential for the intercept (see Groemping 2016 for the maths behind this).

## Value

Function ICFTs returns a list of lists with a component for each projection size considered. Each such component contains the following entries:

| | |
|---|---|
| ICFT | interaction contribution frequency table for the projection size |
| ICs | individual interaction contributions (if requested by option detail) |
| sv2s | squared singular values (if requested by option detail) |
| mean.u2s | squared column means of left-singular vectors (if requested by option detail) |

Function ICFT returns a list with the following components:

| | |
|---|---|
| ICFT | interaction contribution frequency table for the projection size |
| ICs | Average $R^2$ frequency table for the projection size |
| sv2s | squared singular values of the model matrix |
| mean.u2s | squared column means of left-singular vectors in the rotated version (concentrated or even) |
| mm | model matrix of the interaction |

| | |
|---|---|
| u | (left singular vectors corresponding to the rotated version of ICFT (concentrated or even); these do not depend on the coding underlying the model matrix |
| v | (right singular vectors corresponding to the rotated version of ICFT (concentrated or even); these depend on the coding underlying the model matrix |
| c.worst | (v%*%c.worst is the worst case parameter vector for the model matrix mm in terms of bias of the average response for estimation of the intercept caused by the interaction under consideration |

## Warning

The functions have been checked on the types of designs for which they are intended (especially orthogonal arrays produced with oa.design). They may produce meaningless results for some other types of designs.

## Author(s)

Ulrike Groemping

## References

Groemping, U. (2017). An Interaction-Based Decomposition of Generalized Word Counts Suited to Assessing Combinatorial Equivalence of Factorial Designs. *Reports in Mathematics, Physics and Chemistry*, Report 1/2017. http://www1.bht-berlin.de/FB_II/reports/Report-2017-001.pdf, Department II, Berliner Hochschule fuer Technik (formerly Beuth University of Applied Sciences), Berlin.

Groemping, U. (2018). Coding Invariance in Factorial Linear Models and a New Tool for Assessing Combinatorial Equivalence of Factorial Designs. *Journal of Statistical Planning and Inference* **193**, 1-14. doi: 10.1016/j.jspi.2017.07.004.

## See Also

See also GWLP and generalized.word.length

## Examples

```
oa24.bad <- oa.design(L24.2.13.3.1.4.1, columns=c(1,2,14,15))
oa24.good <- oa.design(L24.2.13.3.1.4.1, columns=c(3,10,14,15))
## resolution is III in both cases, but the bad one has more words of length 3
GWLP(oa24.bad)[4:5]
ICFTs(oa24.bad)
ICFTs(oa24.bad, conc=FALSE)
GWLP(oa24.good)[4:5]
ICFTs(oa24.good)
ICFTs(oa24.good, conc=FALSE)
ICFTs(oa24.good, resk.only=FALSE)

ICFT(L18[,c(1,4,6)])
ICFT(L18[,c(1,4,6)], conc=FALSE)
```

`iscube` *Functions to isolate cube points from 2-level fractional factorial design with center and / or star points*

## Description

These functions identify the positions for cube points or star points and can reduce a central composite design to its cube portion (with center points).

## Usage

```
iscube(design, ...)
isstar(design, ...)
pickcube(design, ...)
```

## Arguments

| | |
|---|---|
| `design` | a data frame of class design that contains a 2-level fractional factorial (regular or non-regular) or a central composite design. |
| `...` | currently not used |

## Details

Function `iscube` provides a logical vector that is TRUE for cube points and FALSE for center points and star points. Its purpose is to enable use of simple functions for "clean" 2-level fractional factorials like `MEPlot` or `DanielPlot`.

Function `isstar` provides a logical vector that is TRUE for the star block (including center points) of a central composite design.

Function `pickcube` reduces a central composite design (type `ccd`) to its cube block, including center points. This function is needed, if a CCD has been created in one go, but analyses are already required after conducting the cube portion of the design (and these perhaps even prevent the star portion from being run at all).

## Value

`iscube` and `isstar` each return a logical vector (cf. Details section).

`pickcube` returns a data frame of class design with `type` `FrF2.center` or `FrF2`.

## Warning

For version 0.22-8 of package **DoE.base**, function `iscube` returned a wrong result without warning, when applied to an old version CCD design (before DoE.wrapper, version 0.8-6 of Nov 15 2011). Since version 0.23 of package **DoE.base**, the function works on old designs, except for blocked or replicated versions; for these, an error is thrown.)

### Note

The functions have not been tested for central composite designs for which the cube portion itself is blocked.

### Author(s)

Ulrike Groemping

### References

Montgomery, D.C. (2001). *Design and Analysis of Experiments (5th ed.)*. Wiley, New York.

### See Also

See also as pb, FrF2, ccd.design

### Examples

```
  ## purely technical example, not run because FrF2 not loaded
  ## Not run:
  plan <- FrF2(16,5, factor.names=c("one","two","three","four","five"), ncenter=4)
  iscube(plan)
  plan2 <- ccd.augment(plan)
  iscube(plan2)
  isstar(plan2)
  pickcube(plan2)

## End(Not run)
```

---

## lm and aov method for class design objects

*lm and aov methods for class design objects*

---

### Description

Methods for automatic linear models for data frames of class design

### Usage

```
lm(formula, ...)
## Default S3 method:
lm(formula, data, subset, weights, na.action, method = "qr",
    model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,
    contrasts = NULL, offset, ...)
## S3 method for class 'design'
lm(formula, ..., response=NULL, degree=NULL, FUN=mean,
    use.center=NULL, use.star=NULL, use.dummies=FALSE)
```

```
aov(formula, ...)
## Default S3 method:
aov(formula, data = NULL, projections = FALSE, qr = TRUE,
    contrasts = NULL, ...)
## S3 method for class 'design'
aov(formula, ..., response=NULL, degree=NULL, FUN=mean,
    use.center=FALSE)
## S3 method for class 'lm.design'
print(x, ...)
## S3 method for class 'lm.design'
summary(object, ...)
## S3 method for class 'lm.design'
coef(object, ...)
## S3 method for class 'summary.lm.design'
print(x, ...)
## S3 method for class 'aov.design'
print(x, ...)
## S3 method for class 'aov.design'
summary(object, ...)
## S3 method for class 'summary.aov.design'
print(x, ...)
lm.design
summary.lm.design
aov.design
summary.aov.design
```

## Arguments

| | |
|---|---|
| formula | for the default method, cf. documentation for lm in package **stats**;cr for the class design method, a data frame of S3 class design |
| ... | further arguments to functions lm, print.lm or print.summary.lm |
| response | character string giving the name of the response variable (must be among the responses of x; for wide format repeated measurement or parameter designs, response can also be among the column names of the responselist element of the design.info attribute)<br>OR<br>integer number giving the position of the response in element response.names of attribute design.info<br>For the default NULL, the first available response variable is used; for wide format designs, this is an aggregation of the variables given in first column from the responselist element of the design.info attribute of x. |
| degree | degree for the formula; if NULL, the default for the formula method is used |
| FUN | function for the aggregate.design method; this must be an unquoted function name;<br>This option is relevant for repeated measurement designs and parameter designs in long format only |
| use.center | NULL or logical indicating whether center points are to be used + in the analysis; if NULL, the default is FALSE for pb and FrF2 designs with center points and |

TRUE for ccd designs; the option is irrelevant for all other design types.
FALSE allows usage of simple analysis functions from package FrF2-package (e.g. function IAPlot)

use.star — NULL or logical indicating whether the star portion of a CCD design is to be used in the analysis (ignored for all other types of designs).
The default TRUE analyses the complete design. Specifying FALSE permits interim analyses of the cube portion of a central composite design.

use.dummies — logical indicating whether the error dummies of a Plackett Burman design are to be used in the formula (ignored for all other types of designs).

projections — logical indicating whether the projections should be returned; for orthogonal arrays, these are helpful, as they provide the estimated deviation from the overall average attributed to each particular factor; it is not recommended to use them with unbalanced designs

x — object of class lm or summary.lm,
for lm.default like in lm

object — object of class lm.design created by function lm.design

lm.design — a class that is identical in content to class lm; its purpose is to call a specific print method that provides slightly more detail than the standard printout for linear models

summary.lm.design
a class that is identical in content to class summary.lm; its purpose is to call a specific print method that provides slightly more detail than the standard summary for linear models

data — like in lm

subset — like in lm

weights — like in lm

na.action — like in lm

method — like in lm

model — like in lm

y — like in lm

qr — like in lm

singular.ok — like in lm

contrasts — like in lm

offset — like in lm

## Details

The aov and lm methods for class design conduct a default linear model analysis for data frames of class design that do contain at least one response.

The intention for providing default analyses is to support convenient quick inspections. In many cases, there will be good reasons to customize the analysis, for example by including some but not all effects of a certain degree. Also, it may be statistically more wise to work with mixed models for some types of design. **The default analyses must not be taken as a statistical recommendation!**

The choice of default analyses has been governed by simplicity: It uses fixed effects only and does either main effects models (degree=1, default for pb and oa designs), models with main effects and 2-factor interactions (degree=2, default for most designs) or second order models (that contain quadratic effects in addition to the 2-factor interactions, unchangeable default for designs with quantitative variables). The degree parameter can be used to modify the degree of interactions. If blocks are present, the block main effect is always entered as a fixed effect without interactions.

Designs with center points are per default analysed without the center points; the main reason for this is convenient usage of functions DanielPlot, MEPlot and IAPlot from package **FrF2**. With the use.center option, this default can be changed; in this case, significance of the center point indicator implies that there are one or more quadratic effect(s) in the model.

Designs with repeated measurements (repeat.only=TRUE) and parameter designs of long format are treated by aggregate.design with aggregation function FUN (default: means are calculated) before applying a linear model.

For designs with repeated measurements (repeat.only=TRUE) and parameter designs of wide format, the default is to use the first aggregated response, if the design has been aggregated already. For a so far unaggregated design, the default is to treat the design by aggregate.design, using the function FUN (default: mean) and then use the first response. The defaults can be overridden by specifying response: Here, response can not only be one of the current responses but also a column name of the responselist element of the design.info attribute of the design (i.e. a response name from the long version of the design).

The implementation of the formulae is not done in functions lm.design or aov.design themselves but based on the method for function formula (formula.design).

The print methods prepend the formula and the number of experimental runs underlying the analysis to the default printout. The purpose of this is meaningful output in case a call from inside function lm.design or aov.design (methods for functions lm and aov ) does not reveal enough information, and another pointer that center points have been omitted or repeated measurements aggregated over. The coef method for objects of class lm.design suppresses NA coefficients, i.e. returns valid coefficients only. For aov objects, this is the default anyway.

## Value

The value for the lm functions is a linear model object, exactly like for function lm, except for the added class lm.design in case of the method for class design, and an added list element WholePlotEffects for split plot designs.

The value for the aov functions is an aov object, exactly like for function aov, and an added list element WholePlotEffects for split plot designs.

The value of the summary functions for class lm.design and aov.design respectively is a linear model or aov summary, exactly like documented in summary.lm or summary.aov, except for the added classes summary.lm.design or summary.aov.design, and an added list element WholePlotEffects (for summary.lm.design) or attribute (for summary.aov.design) for split plot designs.

The print functions return NULL; they are used for their side effects only.

## Warning

The generics for lm and aov replace the functions from package **stats**. For normal use, this is not an issue, because their default methods are exactly the functions from package **stats**.

However, when programming on the language (or when using a package that relies on such constructs), you may see unexpected results. For example, `match.call(lm)` returns a different result, depending on whether or not package **DoE.base** is loaded. This can be avoided by explicitly requesting e.g. `match.call(stats::lm)`, which always works in the same way.
Please report any additional issues that you may experience.

## Note

The package is currently subject to intensive development; most key functionality is now included. Some changes to input and output structures may still occur.

## Author(s)

Ulrike Groemping

## See Also

See also the information on class `design` and its formula method `formula.design`

## Examples

```
  oa12 <- oa.design(nlevels=c(2,2,6))
  ## add a few variables to oa12
  responses <- cbind(y=rexp(12),z=runif(12))
  oa12 <- add.response(oa12, responses)
  ## want treatment contrasts rather than the default
  ## polynomial contrasts for the factors
  oa12 <- change.contr(oa12, "contr.treatment")
  linmod.y <- lm(oa12)
  linmod.z <- lm(oa12, response="z")
  linmod.y
  linmod.z
  summary(linmod.y)
  summary(linmod.z)

## examples with aggregation
  plan <- oa.design(nlevels=c(2,6,2), replications=2, repeat.only=TRUE)
  y <- rnorm(24)
  z <- rexp(24)
  plan <- add.response(plan, cbind(y=y,z=z))
  lm(plan)
  lm(plan, response="z")
  lm(plan, FUN=sd)
  ## wide format
  plan <- reptowide(plan)
  plan
  design.info(plan)$responselist
  ## default: aggregate variables for first column of responselist
  lm(plan)
  ## request z variables instead (z is the column name of response list)
  lm(plan, response="z")
  ## force analysis of first z measurement only
```

```
lm(plan, response="z.1")
## use almost all options
## (option use.center can only be used with center point designs
##          from package FrF2)
summary(lm(plan, response="z", degree=2, FUN=sd))
```

---

lowerbound_AR *Function to Calculate a Lower Bound for A_R and Internal Auxiliary Functions*

---

## Description

The functions serve the calculation of lower bounds for the worst case confounding. lowerbound_AR is intended for direct use, lowerbounds and lowerbound_chi2 are internal functions.

## Usage

```
lowerbound_AR(nruns, nlevels, R, crit = "total")
lowerbounds(nruns, nlevels, R)
lowerbound_chi2(nruns, nlevels)
```

## Arguments

| | |
|---|---|
| nruns | positive integer, the number of runs |
| nlevels | vector of positive integers, the numbers of levels for the factors |
| R | positive integer, the resolution of the design; if it is uncertain whether resolution R is feasible, this should be checked by function oa_feasible before applying any of the lower bound functions. |
| crit | "total" or "worst"; if "total", a bound for the overall A_R (sum of the results from lowerbounds) is calculated; otherwise, a bound for the largest individual contribution from an R factor set is calculated |

## Details

Note: if the specified resolution R is not feasible (necessary conditions can be checked with function oa_feasible), any bound(s) returned will be meaningless.

Function lowerbounds provides (integral) bounds on $n^2 A_R$ (with $n$=nruns) according to Groemping and Xu (2014) Theorem 5 for all R factor sets. If the number of runs permits a design with resolution larger than R, the value(s) will be 0. For resolution at least III, the result of function lowerbound_AR is the sum (crit="total") or maximum (crit="worst") of these individual bounds, divided by the square of the number of runs.

For resolution II and crit="total", function lowerbound_chi2 implements the lower bound *B* on $\chi^2$ which was provided in Lemma 2 of Liu and Lin (2009). For supersaturated resolution II designs, this bound is is usually sharper than the one obtained on the basis of Groemping and Xu (2014). Due to the relation between $A_2$ and $\chi^2$ that is stated in Groemping (2017) (summands of $A_2$ are an *n*th

of a $\chi^2$, with $n$=nruns), this bound can be easily transformed into a bound for $A_2$; this relation is also used to slightly sharpen the bound *B* itself: $n^2 \cdot A_2$ must be integral, which implies that *B* can be replaced by ceiling(nruns*B)/nruns, which is applied in function lowerbound_chi2. Function lowerbound_AR increases the lower bound on $A_2$ accordingly, if lowerbound_chi2 provides a sharper bound than the sum of the elements returned by functioni lowerbounds.

## Value

lowerbound_AR returns a lower bound for the number of words of length R (either total or worst case),
lowerbounds returns a vector of lower bounds for individual R factor sets on a different scale (division by nruns^2 needed for transforming this into the contributions to words of length R),
and function lowerbound_chi2 returns a lower bound on the $\chi^2$ value which can be used as a quality criterion for supersaturated designs.

## Author(s)

Ulrike Groemping

## References

Groemping, U. and Xu, H. (2014). Generalized resolution for orthogonal arrays. *The Annals of Statistics* **42**, 918-939.

Groemping, U. (2017). Frequency tables for the coding-invariant quality assessment of factorial designs. *IISE Transactions* **49**, 505-517.

Liu, M.Q. and Lin, D.K.J. (2009). Construction of Optimal Mixed-Level Supersaturated Designs. *Statistica Sinica* **19**, 197-211.

## See Also

See also oa_feasible.

## Examples

```
lowerbound_AR(24, c(2,3,4,6),2)
```

---

# Methods for class design objects

*Methods for class design objects*

---

## Description

Methods for subsetting, aggregating, printing and summarizing class design objects. The formula, lm and plot methods are subject of a separate help page.

## Usage

```
## S3 method for class 'design'
x[i, j, drop.attr = TRUE, drop = FALSE]
## S3 method for class 'design'
print(x, show.order=NULL, group.print=TRUE, std.order=FALSE, ...)
## S3 method for class 'design'
summary(object, brief = NULL, quote = FALSE, ...)
## S3 method for class 'design'
aggregate(x, ...,
    by = NULL, response = NULL, FUN = "mean", postfix = NULL, replace = TRUE)
```

## Arguments

| | |
|---|---|
| x | data frame of S3 class design |
| i | indices for subsetting rows |
| j | indices for subsetting columns |
| drop.attr | logical, controls whether or not attributes are dropped; if TRUE, the result is no longer of class design, and all special design attributes are dropped; otherwise, the design attributes are adjusted to reflect the subsetting result |
| drop | logical that controls dropping of dimensions in the Extract function for data.frame objects, which is called by the method for class design |
| show.order | NULL or logical; if TRUE, the design is printed with run order information; default is TRUE for design types for which this information is helpful (see code for detail), FALSE otherwise |
| group.print | logical, default TRUE; if TRUE, structured designs (blocked and split-plot designs) are printed with intermediate lines at structure breaks; if FALSE, the designs are simply printed as data frames. |
| std.order | logical, default FALSE; if TRUE, the design is printed in standard order rather than in the randomized order. |
| ... | further arguments to functions print, summary, aggregate, contrasts |
| object | data frame of S3 class design, like argument design |
| brief | NULL or logical; TRUE requests a printout of the design at the end of the summary output, FALSE suppresses such a printout. If brief = NULL (the default), the summary method prints the design object if it has up to 40 rows and up to 20 columns. |
| quote | logical; TRUE requests quoting strings in print parts of the output, FALSE suppresses quotes. |
| by | by variables for the data frame method of function aggregate, needed if x is not a wide design for which the special method for class design is intended |
| response | used for wide format designs only;<br>if NULL, all responses of the design are aggregated; specify names of selected responses (column names of the responselist element of the design.info attribute) for restricting the responses that are treated |

| | |
|---|---|
| FUN | a function to be used for aggregation, the default is "mean"; can be used like the FUN argument to apply |
| postfix | NULL implies postfixing the response name with (a character version of) FUN; a character string can be given instead for a user-defined postfix |
| replace | logical that decides whether an existing variable of the given name is to be replaced; the default is TRUE for convenience reasons. WARNING: If custom variables other than aggregation variables are added to wide format designs, it is recommended to use variables names that are not likely to be generated by this function. |

## Details

Items of class design are data frames with attributes, that have been created for conducting experiments. Apart from the methods documented here, separate files document the methods formula.design and plot.design.

The extractor method subsets the design, taking care of the attributes accordingly (cf. the value section). Subsetting can also handle replication in a limited way, although this is not first choice. Repeated measurements can be added to a design that has no proper replications, and proper replications can be added to a design that has no repeated measurements.

The method for print displays the design. Per default, the design is printed in the actual run order, and run order information is shown for designs with special structure (blocked, replicated). Optionally, the design can be printed in standard order, which may be useful for comparing to other designs or for getting a clearer idea about the structure of smaller designs.

The method for summary provides design-specific information - some further development may still be expected. If a standard data frame summary is desired, explicitly use function summary.data.frame instead of summary.

The method for aggregate provides aggregation utilities for wide format designs and links back to the method for data frames for designs that are not of wide format. If a wide format design is to be treated with the aggregate method for data frames, aggregate.data.frame must be used explicitly. This method calculates a mean, standard deviation or SN ratio from the individual responses (which can be repeated measurements or outer array runs from a Taguchi parameter design).

## Value

| | |
|---|---|
| extractor | The extractor function returns a class design object with modified attributes or a data frame without special attributes, depending on the situation.<br>If j is given, the function always returns a data frame without special attributes, even if drop.attr=FALSE or j=1:ncol(design).<br>If only i is given, the default option drop.attr=TRUE also returns a data frame without attributes.<br>Exception: Even for drop.attr=TRUE, if i is a permutation of the row numbers or a logical vector with all elements TRUE, the attributes are preserved, and attributes run.order and desnum are reordered along with the design, if necessary.<br>If drop.attr=FALSE and j is empty, the function returns an object of class design with rows of attributes run.order and desnum selected in line with those |

of the design itself. In this case, the new `design.info` attribute is a list with entries

**type** resolving to "subset of design",

**subset.rows** a numeric or logical vector with the selected rows, and

**orig.design.info** which contains the original `design.info` attribute.

The `print` and `summary` methods are called for their side effects and return `NULL`.

The method for `aggregate` returns the input wide format design with one or more additional response columns and the `response.names` element of the `design.info` attribute changed to only include the newly-added responses.

## Note

The package is currently subject to intensive development; most key functionality is now included. Some changes to input and output structures may still occur.

## Author(s)

Ulrike Groemping

## See Also

See also the following functions known to produce objects of class `design`: `FrF2`, `pb`, `fac.design`, `oa.design`.
See also the following further methods for class `design` objects: `formula.design`, `lm.design`, `plot.design`.
Function `plot.design` from package graphics works on data frames with R factors as explanatory variables, if a numeric response is available; this function is invoked by method `plot.design` from this package, where appropriate.

## Examples

```
oa12 <- oa.design(nlevels=c(2,2,6))
#### Examples for extractor function
  ## subsetting to half the runs drops all attributes per default
  oa12[1:6,]
  ## keep the attributes (usually not reasonable, but ...)
  oa12[1:6, drop.attr=FALSE]
  ## reshuffling a design
  ## (re-)randomize
  oa12[sample(12),]
  ## add repeated measurements
  oa12[rep(1:12,each=3),]
  ## add a proper replication
  ## (does not work for blocked designs)
  oa12[c(sample(12),sample(12)),]
  ## subsetting and rbinding to loose also contrasts of factors
  str(rbind(oa12[1:2,],oa12[3:12]))
  ## keeping all non-design-related attributes like the contrasts
  str(undesign(oa12))
```

```
#### Examples print and summary
  ## rename factors and relabel levels of first two factors
  namen <- c(rep(list(c("current","new")),2),list(""))
  names(namen) <- c("First.Factor", "Second.Factor", "Third.Factor")
  factor.names(oa12) <- namen
  oa12   ### printed with the print method!

  ## add a few variables to oa12
  responses <- cbind(temp=sample(23:34),y1=rexp(12),y2=runif(12))
  oa12 <- add.response(oa12, responses)
  response.names(oa12)
  ## temp (for temperature) is not meant to be a response
  ## --> drop it from responselist but not from data
  response.names(oa12) <- c("y1","y2")

## print design
  oa12
## look at design-specific summary
  summary(oa12)
## look at data frame style summary instead
  summary.data.frame(oa12)

## aggregation examples
  plan <- oa.design(nlevels=c(2,6,2), replications=2, repeat.only=TRUE)
  y <- rnorm(24)
  z <- rexp(24)
  plan <- add.response(plan, cbind(y=y,z=z))
  plan <- reptowide(plan)
  plan.mean <- aggregate(plan)
  plan.mean
  aggregate(plan, response="z")
  aggregate(plan, FUN=sd)
  aggregate(plan, FUN = function(obj) max(obj) - min(obj), postfix="range")
  ## several aggregates: add standard deviations to plan with means
  plan.mean.sd <- aggregate(plan.mean, FUN=sd)
  plan.mean.sd
  response.names(plan.mean.sd)
  ## change response.names element of design.info back to y.mean and z.mean
  ## may be needed for automatic analysis routines that have not been
  ## created yet
  plan.mean.sd <- aggregate(plan.mean.sd, FUN=mean)
  plan.mean.sd
  response.names(plan.mean.sd)
```

---

`oa.design` *Function for accessing orthogonal arrays*

---

## Description

Function for accessing orthogonal arrays, allowing limited optimal allocation of columns

## Usage

```
oa.design(ID=NULL, nruns=NULL, nfactors=NULL, nlevels=NULL,
      factor.names = if (!is.null(nfactors)) {
        if (nfactors <= 50) Letters[1:nfactors]
           else paste("F", 1:nfactors, sep = "")}
        else NULL,
        columns="order",
        replications=1, repeat.only=FALSE,
        randomize=TRUE, seed=NULL, min.residual.df=0, levordold = FALSE)
origin(ID)
```

## Arguments

| | |
|---|---|
| ID | orthogonal array to be used; must be given as the name without quotes (e.g. L12.2.2.6.1); available names can be looked at using function show.oas; furthermore, L18, L36 and L54 for the respective Taguchi arrays can be used.<br>Users can also specify names of their own designs here (cf. details).<br>ID must be of class oa. If omitted, ID is automatically determined based on nlevels or factor.names. |
| nruns | minimum number of runs to be used, can be omitted if obvious from ID or if the smallest possible array is to be found |
| nfactors | number of factors; only needed if nlevels is a single number and factor.names is omitted; can otherwise be determined from length of factor.names, nlevels or column |
| nlevels | number(s) of levels, vector with nfactors entries or single number; can be omitted, if obvious from factor.names or from ID,<br>or if ID and columns are given<br>or if all columns of ID are to be used with default factor names and levels;<br>can be a single number if nfactors is known directly or as length of factor.names |
| factor.names | a character vector of nfactors factor names or a list with nfactors elements;<br>if the list is named, list names represent factor names, otherwise default factor names are used;<br>the elements of the list are<br>EITHER vectors of appropriate length (corresponding to nlevels) with factor levels for the respective factor<br>OR empty strings;<br>Default factor names are the first elements of the character vector Letters, or the factors position numbers preceded by capital F in case of more than 50 factors. Default factor levels are the numbers from 1 to the number of levels for each factor. |
| columns | EITHER<br>a vector of column numbers referring to columns of design ID, assigning a specific column of the array to each factor;<br>this can only be specified, if ID is also given;<br>OR<br>a string that defines the degree of optimization requested in terms of column allocation (cf. section "Details"): choices are "order", "min3", "min34", "min3.rela", |

"min34.rela", "minRPFT" or "minRelProjAberr".
For resource reasons, the default is "order", but smaller designs can sometimes be substantially improved with other choices. Cf. the "Details" section for the meaning of the character string specifications for columns. Column optimization can be computationally intensive. If it cannot be accomplished with the given ressources, a warning is issued, and an unoptimized design is returned.
Some of the optimization methods have just been proposed, and there is little experience with them. It is strongly recommended to always check the properties of the design w.r.t. suitability for the planned experiment BEFORE starting expensive investments.

replications — the number of replications of the array, the setting of repeat.only determines, whether these are real replications or repeated measurements only. Note that replications are not considered for accomodation of min.residual.df residual degrees of freedom, unless a full factorial is used.

repeat.only — default FALSE implies real replications, TRUE implies repeated measurements only

randomize — logical indicating whether the run order is to be randomized ?

seed — integer seed for the random number generator
In R version 3.6.0 and later, the default behavior of function sample has changed. If you work in a new (i.e., >= 3.6.-0) R version and want to reproduce a randomized design from an earlier R version (before 3.6.0), you have to change the RNGkind setting by
RNGkind(sample.kind="Rounding")
before running function oa.design.
It is recommended to change the setting back to the new recommended way afterwards:
RNGkind(sample.kind="default")
For an example, see the documentation of the example data set VSGFS.

min.residual.df — minimum number of residual degrees of freedom;
Note: function oa.design does not count replications specified with option replications in determining residual degrees of freedom for min.resid.df.

levordold — logical indicating whether or not old (=pre version 0.27) level ordering should be used;
defaults to FALSE, which implies that levels are ordered as indicated in factor.names; in the old ordering, levels were automatically reordered by the as.factor function, which is usually undesirable, but may be desired for reproducing designs created with earlier versions

## Details

Package **DoE.base** is described in Groemping (2018). The paper also has detailed material on function oa.design.

Function oa.design assigns factors to the columns of orthogonal arrays that are available within package **DoE.base** or are provided by the user. The available arrays and their properties are listed in the data frame oacat and can be systematically searched for using function show.oas. The design

names also indicate the number of runs and the numbers of factors for each number of levels, e.g. `L18.3.6.6.1` is an 18 run design with six factors in 3 levels (`3.6`) and one factor in 6 levels (`6.1`).

`oa` is the S3 class used for orthogonal arrays. Objects of class `oa` should at least have the attribute `origin`, an attribute `comment` should be used for additional information.

Users can define their own orthogonal arrays and hand them to `oa.design` with parameter `ID`.
Requirements for the arrays:

- Factor levels must be coded as numbers from 1 to number of levels.
- The array must be of classes `oa` and `matrix`
  (If your array is a matrix named `foo`, you can simply assign it class `oa` by the command `class(foo) <- c("oa","matrix")`, see also last example.)
- The array should have an attribute `origin`.
- The array can have an attribute `comment`;
  this should be used for mentioning specific properties, e.g. for the `L18.2.1.3.7` that the interaction of the first two factors can be estimated.

Users are encouraged to send additional arrays to the package maintainer. The requirements for these are the same as listed above, with attribute `origin` being a MUST in this case. (See the last example for how to assign an attribute.)
For relatively small important applications, creation of a tailor-made array of class `oa` can be attempted with package **DoE.MIParray**, which uses mixed integer optimization for creating a design from scratch (see Groemping and Fontana 2019 for the algorithm behind that approach).

The data frame `oacat` lists the orthogonal arrays from Warren Kuhfelds collection of "parent" and "child" arrays. The parent arrays, plus a few additional arrays, are directly exported from the package namespace. The child arrays from Kuhfelds collection can be constructed from these, using the replacement instructions provided in the variable `lineage` of `oacat`. The last example below indicates how a child array can be created manually, and compares this to the automatically created array.
(A lot more than just the child arrays could be obtained from these arrays by implementing a functionality similar to the market research macros available in SAS; presumably, this topic will not be addressed soon, as it will involve a substantial amount of work.)

Furthermore, there are stronger arrays (at least resolution IV) in the catalogue `oacat3`. Since version 1.1, function `oa.design` uses the stronger arrays, where possible.

If no specific orthogonal array is specified and function `oa.design` does not find an orthogonal array that meets the specified requirements, `oa.design` returns a full factorial, replicated for enough residual degrees of freedom, if necessary. If `oa.design` has not found an array smaller than the full factorial, it is absolutely possibly that a smaller array does exist nevertheless. It may be worth while checking with oacat whether an appropriate smaller array can be found by combining some of the parent arrays listed there (looking for a design with a few factors in 5 runs, you may e.g. call `oacat[oacat$n5>0,]$name` in order to see the names of more promising candidate arrays for combination, or you may also want to look up arrays with `n25>0` subsequently.

With version 0.9-18 of the package, the possibility for an automatic allocation of columns for improved design performance was implemented. With version 0.23, this approach has been sped up and extended to properly cover relative projection aberration according to Groemping (2011) with

and without step (b) (see below) (the previous choice "maxGR.min34" was modified and renamed to "minRelProjAberr").
Because of performance reasons, and because of a lack of a clear best default, optimum column allocation is not switched on per default. However, with the default column order from left to right, the package always issues a warning to remind users that an automatic unoptimized design can be quite far from ideal. If optimization is activated, the first step is selection of an array, either explicitly by the user (option ID) or automatically (unoptimized) according to the required combination of factors. Within that array, the following choices for the column option are on offer:

**"order"** the default choice; allocates factors from left to right, which is what most software does (but what is not necessarily good, see also the example section)

"min3" **recommended**, if "min34" is not affordable; aliasing between main effects and 2-factor interactions is kept to a minimal degree, minimizing the number of generalized words of length 3 according to Xu and Wu (2001)

**"min3.rela"** the same approach is taken, but with *relative* number of generalized words according to Groemping (2011)

**"min34" recommended**, if affordable; beware the time demand; this requests that the number of words of generalized length 4 is also minimized.

**"min34.rela"** again takes the same approach, but with *relative* number of generalized words according to Groemping (2011)

**"minRPFT"** minimizes the relative projection frequency table, applying the approach according to Groemping (2011) without step (b) (see next entry).

**"minRelProjAberr"** applies minimum relative projection aberration according to Groemping (2011) ((a): maximize generalized resolution, (b): minimize total relative number of shortest words, (c) rank designs according to relative projection frequency table (obtainable with P3.3 or P4.4, depending on resolution) and (d) resolve ties by looking at absolute number of length 4 words in case of resolution III).

WARNING: Usually, it is recommended to investigate the properties of a design automatically created by function oa.design before starting experimentation. While all designs can estimate main effects *in the absence of interactions*, the presence of interactions may render some designs useless or even dangerous. Deliberate choice of columns different from the default may improve a design (see example section)!

Mathematical comment on the expansion example: There are 720 different ways to expand the unique L18.3.6.6.1 into an L18.2.1.3.7, depending on which row of the replacement design nest.des is assigned to which level of the 6 level factor; for qualitative factors, 60 of these are potentially non-isomorphic (divide 720 by the 2 * 3! ways of permuting levels within a factor; there are more possibly different arrays for quantitative 3 level factors, since arbitrary relabelling of the levels is no longer isomorphic). According to Eric Schoen (personal communication), for this particular case, all the resulting children are isomorphic to each other and are also isomorphic to the Taguchi L18. To see isomorphism of two designs is not easy; in the example, nest.des has been prepared such that it is easy to see isomorphism of the resulting child to the Taguchi L18: L18 is reproduced by assigning the first row of nest.des to level 1 etc., except for a swap of columns G and H.

## Value

`oa.design` returns a data frame of S3 class `design` with attributes attached.

In the data frame itself, the experimental factors are all stored as R factors.
For factors with 2 levels, `contr.FrF2` contrasts (-1 / +1) are used.
For factors with more than 2 numerical levels, polynomial contrasts are used (i.e. analyses will per default use orthogonal polynomials).
For factors with more than 2 categorical levels, the default contrasts are used.
Future versions will most likely allow more user control about the type of contrasts to be used.

The `desnum` and `run.order` attributes of class `design` are as usual. In the `design.info` attribute, the following elements are specific for this type of designs:

| | |
|---|---|
| `type` | is `oa` (unless no special orthogonal array is found, in which case a full factorial is created instead, cf. `fac.design` for its `design.info` attribute), |
| `nlevels` | vector containing the number of levels for each factor |
| `generating.oa` | contains information on the generating orthogonal array, |
| `selected.columns` | contains information, which column of the orthogonal array underlies which factor, |
| `origin` | contains the respective attribute of the orthogonal array, |
| `comment` | contains the respective attribute of the orthogonal array, |
| `residual.df` | contains the requested residual degrees of freedom for a main effects model. |

Other information is generic, like documented for class `design`.

Function `origin` returns the origin attribute of the orthogonal array `ID`, functions `comment` and `"comment<-"` from package **base** return and set the comment attribute.

## Warning

Since version 1.1 of the package, strength 3 arrays are automatically used, if available. This changes the behavior of function `oa.design` for situations for requests with a combination of `nruns` and `nlevels` for which a strength 3 array exists in `oacat3`. If the old behavior is required for reproducing a previously-created array, it is possible to set `oacat3` to `NULL` by the command `assignInNamespace("oacat3", NULL, pos="package:DoE.base")`; this temporary replacement of `oacat3` with `NULL` remains in effect for the current R session; detaching it (with namespace unloading) and reloading is possible but can also go wrong; therefore, it is recommended to only use the above technique if you are prepared to restart the R session before using the original version of `oacat3`.

Since R version 3.6.0, the behavior of function `sample` has changed (correction of a biased previous behavior that should not be relevant for the randomization of designs). For reproducing a randomized design that was produced with an earlier R version, please follow the steps described with the argument `seed`.

## Note

This package is still under development. Suggestions and bug reports are welcome.

## Author(s)

Ulrike Groemping

## References

Groemping, U. (2011). Relative projection frequency tables for orthogonal arrays. Report 1/2011, *Reports in Mathematics, Physics and Chemistry* http://www1.bht-berlin.de/FB_II/reports/welcome.htm, Department II, Berliner Hochschule fuer Technik (formerly Beuth University of Applied Sciences), Berlin.

Groemping, U. (2018). R Package DoE.base for Factorial Designs. *Journal of Statistical Software* **85**(5), 1–41.

Hedayat, A.S., Sloane, N.J.A. and Stufken, J. (1999) *Orthogonal Arrays: Theory and Applications*, Springer, New York.

Kuhfeld, W. (2009). Orthogonal arrays. Website courtesy of SAS Institute https://support.sas.com/techsup/technote/ts723b.pdf and references therein.

Schoen, E. (2009). All orthogonal arrays with 18 runs. *Quality and Reliability Engineering International* **25**, 467–480.

Xu, H.-Q. and Wu, C.F.J. (2001). Generalized minimum aberration for asymmetrical fractional factorial designs. *Annals of Statistics* **29**, 1066–1077.

## See Also

See Also FrF2, fac.design, pb

## Examples

```
## smallest available array for 6 factors with 3 levels each
oa.design(nfactors=6, nlevels=3)
## level combination for which only a full factorial is (currently) found
oa.design(nlevels=c(4,3,3,2))
## array requested via factor.names
oa.design(factor.names=list(one=c("a","b","c"), two=c(125,275),
   three=c("old","new"), four=c(-1,1), five=c("min","medium","max")))
## array requested via character factor.names and nlevels
  ## (with a little German lesson for one two three four five)
oa.design(factor.names=c("eins","zwei","drei","vier","fuenf"), nlevels=c(2,2,2,3,7))
## array requested via explicit name, Taguchi L18
oa.design(ID=L18)
## array requested via explicit name, with column selection
oa.design(ID=L18.3.6.6.1,columns=c(2,3,7))
## array requested with nruns, not very reasonable
oa.design(nruns=12, nfactors=3, nlevels=2)
## array requested with min.residual.df
oa.design(nfactors=3, nlevels=2, min.residual.df=12)

## examples showing alias structures and their improvment with option columns
plan <- oa.design(nfactors=6,nlevels=3)
plan
   ## generalized word length pattern
```

```
   length3(plan)
   ## length3 (first element of GWP) can be slightly improved by columns="min3"
   plan <- oa.design(nfactors=6,nlevels=3,columns="min3")
   summary(plan)  ## the first 3-level column of the array is not used
   length3(plan)
plan <- oa.design(nlevels=c(2,2,2,6))
   length3(plan)
plan.opt <- oa.design(nlevels=c(2,2,2,6),columns="min3") ## substantial improvement
   length3(plan.opt)
   length4(plan.opt)
## visualize practical relevance of improvement:
   ## for optimal plan, all 3-dimensional projections are full factorials
plot(plan, select=1:3)
plot(plan, select=c(1,2,4))
plot(plan, select=c(1,3,4))
plot(plan, select=2:4)
plot(plan.opt, select=1:3)
plot(plan.opt, select=c(1,2,4))
plot(plan.opt, select=c(1,3,4))
plot(plan.opt, select=2:4)


## The last example:
## generate an orthogonal array equivalent to Taguchi's L18
## by combining L18.3.6.6.1 with a full factorial in 2 and 3 levels

show.oas(nruns=18, parents.only=FALSE)
     ## lineage entry leads the way:
         ## start from L18.3.6.6.1
         ## insert L6.2.1.3.1 for the 6 level factor
## prepare the parent
 parent.des <- L18.3.6.6.1
 colnames(parent.des) <- c(Letters[3:9])
     ## new columns will become A and B
## 6-level design can be created by fac.design or expand.grid or cbind
 nest.des <- as.matrix(expand.grid(1:3,1:2))[c(1:3,5,6,4),c(2,1)]
      ## want first column to change most slowly
      ## want resulting design to be easily transformable into Taguchi L18
      ## see mathematical comments in section Details
 colnames(nest.des) <- c("A","B")
## do the expansion (see mathematical comments in section Details)
## using function expansive.replace
L18.2.1.3.7.manual <- expansive.replace(parent.des, nest.des)[,c(7:8,1:6)]
L18.2.1.3.7.manual <- L18.2.1.3.7.manual[ord(L18.2.1.3.7.manual),]  ## sort array
    rownames(L18.2.1.3.7.manual) <- 1:18
      ## (ordering is not necessary, just **tidy**)
## prepare for using it with function oa.design
      ## note: function expansive.replace creates a matrix of class "oa"
      ## rearranging the columns removed that class and makes it necessary
      ##    to add the class again for using the array in DoE.base
attr(L18.2.1.3.7.manual, "origin") <-
    c(show.oas(name="L18.2.1.3.7", parents.only=FALSE,show=0)$lineage,
      "unconventional order")
```

```
class(L18.2.1.3.7.manual) <- c("oa", "matrix")
comment(L18.2.1.3.7.manual) <- "Interaction of first two factors estimable"
   ## indicates that first two factors are full factorial from 6-level factor
origin(L18.2.1.3.7.manual)
comment(L18.2.1.3.7.manual)
L18  ## Taguchi array
L18.2.1.3.7.manual  ## manually expanded array
oa.design(L18.2.1.3.7, randomize=FALSE)
      ## automatically expanded array
P3.3(L18.2.1.3.7.manual)  ## length 3 pattern of 3 factor projections
                ## this also identifies the array as isomorphic to L18
                ## according to Schoen 2009
## the array can now be used in oa.design, like the built-in arrays
oa.design(ID=L18.2.1.3.7.manual,nfactors=7,nlevels=3)
```

---

oacat *Data Frames That List Available Orthogonal Arrays*

---

## Description

These data frames hold the lists of available orthogonal arrays, except for a few structurally equivalent additional arrays known as Taguchi arrays (L18, L36, L54). Arrays in in oacat are mostly from the Kuhfeld collection, those in oacat3 from some other sources.

## Usage

```
oacat
oacat3
```

## Details

The data frames hold a list of orthogonal arrays, as described in Section "value". Inspection of these arrays can be most easily done with function show.oas. Some of the listed arrays are directly accessible through their names ("parent" arrays, also listed under arrays) or are full factorials the construction of which is obvious. Others can be constructed as "child" arrays from the parent and full factorial arrays, using a so-called lineage which is also included as a column in data frame oacat. Most of the listed arrays have been taken from Kuhfeld 2009. Exceptions: The three arrays L128.2.15.8.1, L256.2.19 and L2048.2.63) have been taken from Mee 2009; these are irregular resolution IV or V arrays for which all main effects can be orthogonally estimated even in the presence of interactions, or even all 2fis can be orthogonally estimated, provided there are no higher order effects.

Note that most of the arrays in oacat, per default, are guaranteed to orthogonally estimate all main effects, **provided all higher order effects are negligible** (again, the Mee arrays are an exception). This can be a very severe limitation, of course, and arbitrary strong biases can distort the estimates even of main effects, if this assumption is violated. It is therefore strongly recommended to inspect the quality of an orthogonal array quite closely before deciding to use it for experimentation. Some functions for inspecting arrays are provided in the package (cf. generalized.word.length).

The data frame oacat3 contains stronger arrays that have at least the main effects unconfounded with two-factor interactions. If only these are of interest, function show.oas can be restricted to strong arrays by option Rgt3=TRUE. Function oa.design will use a strong array, if possible. Since package version 1.2, oacat3 contains arrays that were obtained via expansive replacement (indicated in the lineage column). It is important to note that this automatic replacement is not optimized in any way; in some cases it may be worthwhile to check whether a better array can be produced with different level choices or by expanding not the first but a different column of the parent array (for an example, see function expansive.replace); this is not automatically checked and can only be done by the user.

## Value

The data frames contain the columns name, nruns, lineage and further columns n2 to n72; furthermore, some columns with calculated metrics are included. name holds the name of the array, nruns its number of runs, and lineage the way the array can be constructed from other arrays, if applicable. The columns n2 to n72 each contain the number of factors with the respective number of levels.

The logical columns ff, regular.strict and regular indicate a full factorial and a regular design in the strict or weak sense, respectively (strict: all ARFT entries are 0 or 1, defined as "$R^2$ regular" in Groemping and Bailey (2016); weak: all SCFT entries are 0 or 1, defined as "CC regular" in Groemping and Bailey (2016)). For $R^2$ regularity, it suffices to check all full resolution factor sets, i.e., sets of j factors with resolution j; for CC regularity, this is conjectured to be also true. The entries in column regular are based on that conjecture (and for some larger designs, even those checks were not completed); thus, designs denominated as CC regular might prove otherwise if the conjecture proves wrong, and for larger designs also for unchecked full resolution factor sets of higher dimensions).

Column SCones contains the number of worst case (=1) squared canonical correlations for the number of R factor subsets, with R the resolution; if this number is 0, main effects can be considered to have partial confounding only with any interactions of up to R-1 factors. GR, GRind, maxAR and maxSC contain the generalized resolution in two versions, the maximum average $R^2$ and the maximum squared canonical correlation.

dfe contains the error degrees of freedom of a main effects model, if all columns of the array are populated; if this is 0, the design is saturated. A3 to A8 contain the numbers of words of lengths 3 to 8. More information on these metrics can be found in generalized.word.length and the literature therein.

The design names also indicate the number of runs and the numbers of factors: The first portion of each array name (starting with L) indicates the number of runs, each subsequent pair of numbers indicates a number of levels together with the frequency with which it occurs. For example, L18.2.1.3.7 is an 18 run design with one factor with 2 levels and seven factors with 3 levels each.

The columns gmarule and sgmarule refer to the implementation of known rules from the literature that certain subsets of array columns have generalized minimum aberration (Butler 2005); if such a subset is requested, there is no message of caution even if the array columns are used with column="order" instead of optimizing the selection. Currently, only the rules from Butler (2005) are implemented; hopefully, more rules will be added in the future.

The column lineage deserves particular attention: it is an empty string, if the design is directly available and can be accessed via its name, or if the design is a full factorial (e.g. L6.2.1.3.1). Otherwise, the lineage entry is structured as follows: It starts with the specification of a parent

array, given as `levels1~no of factors; levels2~no of factors;`. After a colon, there are one or more replacements, each enclosed in brackets; within each pair of brackets, the left-hand side of the exclamation mark shows the to-be-replaced factor, the right-hand side the replacement array that has to be used for replacing the levels of such a factor one or more times. For example, the lineage for `L18.2.1.3.7` is `3~6;6~1;:(6~1!2~1;3~1;)`, which means that the parent array in 18 runs with six 3 level factors and one 6 level factor has to be used, and the 6 level factor has to be replaced with the full factorial with one 2 level factor and one 3 level factor.

## Warning

For designs with only 2-level factors, it is usually more wise to use package **FrF2**. Exceptions: The three arrays by Mee (2009; cf. section "Details" above) are very useful for 2-level factors.

Many of the orthogonal arrays from `oacat`, especially when using all columns for experimentation, are guaranteed to orthogonally estimate all main effects, **provided all higher order effects are negligible**.

Make sure you understand the implications of using an orthogonal main effects design for experimentation. In particular, for some designs there is a very severe risk of obtaining biased main effect estimates, if there are some interactions between experimental factors. The documentation for `generalized.word.length` and examples section below that illustrate this remark. Cf. also the instructions in section "Details").

## Author(s)

Ulrike Groemping, with contributions by Boyko Amarov

## References

Agrawal, V. and Dey, A. (1983). Orthogonal resolution IV designs for some asymmetrical factorials. *Technometrics* **25**, 197–199.

Brouwer, A. Small mixed fractional factorial designs of strength 3. https://www.win.tue.nl/~aeb/codes/oa/3oa.html#toc1 accessed March 1 2016

Brouwer, A., Cohen, A.M. and Nguyen, M.V.M. (2006). Orthogonal arrays of strength 3 and small run sizes. *Journal of Statistical Planning and Inference* **136**, 3268–3280.

Butler, N.A. (2005). Generalised minimum aberration construction results for symmetrical orthogonal arrays. *Biometrika* **92**, 485 – 491.

Eendebak, P. and Schoen, E. Complete Series of Orthogonal Arrays. http://www.pietereendebak.nl/oapackage/series.html accessed March 1 2016

Groemping, U. and Bailey, R.A. (2016). Regular fractions of factorial arrays. In: *mODa 11 – Advances in Model-Oriented Design and Analysis*. Cham: Springer International Publishing.

Groemping, U. and Fontana, R. (2019). An Algorithm for Generating Good Mixed Level Factorial Designs. *Computational Statistics and Data Analysis* **137**, 101–114.

Kuhfeld, W. (2009). Orthogonal arrays. Website courtesy of SAS Institute https://support.sas.com/techsup/technote/ts723b.pdf and references therein.

Mee, R. (2009). *A Comprehensive Guide to Factorial Two-Level Experimentation*. New York: Springer.

Nguyen, M.V.M. (2005). *Journal of Statistical Planning and Inference* **138**, 220–233.

Nguyen, M.V.M. (2008). Some new constructions of strength 3 mixed orthogonal arrays. *Journal of Statistical Planning and Inference* **138**, 220–233.

Sloane, N. Orthogonal Arrays. http://neilsloane.com/oadir/ accessed March 1 2016

## See Also

oa.design for using the designs from oacat in design creation
show.oas for inspecting the available arrays from oacat
generalized.word.length for inspection functions for array properties
arrays for a list of orthogonal arrays which are directly accessible within the package

## Examples

```
head(oacat)

sapply(oacat3$name[which(oacat3$lineage=="")],
     function(nn) unlist(attributes(get(nn))[c("origin", "comment")]))
```

---

# oa_feasible *Function to Check Whether an Array of Specified Strength Might Exist*

---

## Description

The function checks necessary conditions for the existence of an array of specified strength

## Usage

```
oa_feasible(nruns, nlevels, strength = 2, verbose=TRUE, returnbound=FALSE)
```

## Arguments

| | |
|---|---|
| nruns | positive integer, number of rows |
| nlevels | vector of positive integers: its length determines the number of columns, the elements determine the numbers of levels for each column |
| strength | positive integer (default 2), not larger than the length of nlevels requested strength of array; 1+strength is the resolution |
| verbose | logical; if TRUE, reason for outcome is printed |
| returnbound | logical; if TRUE, the function returns a lower bound for the number of runs needed instead of a logical |

## Details

The function uses several known bounds and necessary divisibility requirements on `nruns` for checking *potential* feasibility of an array of the requested strength. It is checked that `nruns` is a multiple of the LCM of the run sizes of unreplicated full factorials of all sets of `strength` factors and that Rao's bound is fulfilled (the simplest one for strength 2 arrays being that `nruns` is larger than the sum of the main effect degrees of freedom; formulae available in Hedayat et al. 1999 Theorem 2.1 for pure levels and Diestelkamp 2004 Theorem 3.1 for mixed levels). For pure level designs, the Bush bounds and Bose/Bush bounds are implemented (Hedayat et al., Theorems 2.8, 2.11 and 2.19). Furthermore, Bierbrauer's bound (Diestelkamp 2004 Theorems 2.1 and 2.2) is implemented for pure and mixed level designs; note that the mixed level formula has been applied for large `strength` values only, because the proof of Diestelkamp is valid only for these (contrary to what is claimed in the paper). For pure 2-level-designs, the bound from Bierbrauer et al. (1999) is also implemented. All these are necessary but not a sufficient conditions for the existence of an orthogonal array of the requested strength.

The implemented bounds have been verified against selected scenarii from Tables 12.1 to 12.3 of Hedayat, Sloane and Stufken 1999. These tables detect further infeasibilities, since they incorporate detailed research results for specific scenarii, contrary to this function which only checks straightforward explicit bounds. Another resource for checking feasibility of symmetric OAs (i.e. OAs with the same number of levels for all factors) is the website `http://mint.sbg.ac.at/`.

## Value

A logical or an integer number.
For `returnbound=FALSE` (default), a logical is returned: if FALSE, an OA is infeasible; if TRUE, an OA *might* be feasible.
For `returnbound=TRUE`, an integer is returned: a *lower bound* for the number of runs needed for an OA with the requested strength.

## Author(s)

Ulrike Groemping

## References

Bierbrauer, J., Gopalakrishnan, K. and Stinson, D.R. (1999). Orthogonal Arrays, Resilient Functions, Error Correcting Codes and Linear Programming Bounds. Working paper (expanded and revised version of a published extended abstract of the same authors). `https://pages.mtu.edu/~jbierbra/`.

Diestelkamp, W. (2004). Parameter inequalities for orthogonal arrays with mixed levels. *Designs, Codes and Cryptography* **33**, 187-197.

Hedayat, S., Sloane, N.J.A. and Stufken, J. (1999). Orthogonal Arrays. Springer, New York.

## See Also

See also function `show.oas` of package **DoE.base** for orthogonal arrays catalogued in that package.

## Examples

```
## strength 2 equal to resolution 3 is the default
## pure level examples (function checks criteria in the order listed here)
oa_feasible(51, rep(5,7))
   ## nruns not divisible by 5^2
oa_feasible(1024, rep(2,14), strength=7)
   ## violates Bierbrauer et al.s bound for 2-level
oa_feasible(6561, rep(3,11), strength=8)
   ## violates Bierbrauer's bound for pure level
oa_feasible(25, rep(5,7))
   ## violates Rao's bound for pure level
oa_feasible(256,rep(4,7), 4)
   ## violates Bush bound (checked for pure level only)
oa_feasible(54, rep(3,26))
   ## violates Bose/Bush bound (checked for pure level only)
oa_feasible(25, rep(5, 12), strength = 1)
   ## feasible; but do not try to optimize (5^12 integer variables!!!)
oa_feasible(243, rep(3,11), strength = 4)
   ## strength 4 design that strictly attains the Rao bound for pure level

## mixed level examples (function checks criteria in the order listed here)
oa_feasible(25, c(rep(5,6),4))
   ## too few df for main effects (special case of Rao's bound)
oa_feasible(100, c(rep(5,6),4), 5)
   ## violates Diestelkamps mixed level version of Bierbrauer's bound
               ## (also violates Rao's bound, but this is checked earlier)
oa_feasible(100, c(rep(5,7),4), 3)
   ## violates Rao's bound for mixed level, strength 3
oa_feasible(100, c(rep(5,7),4), 4)
   ## violates Rao's bound for mixed level, even strength
oa_feasible(100, c(rep(5,7),4), 5)
   ## violates Rao's bound for mixed level, general odd strength
oa_feasible(50, c(2,rep(5,12)))
   ## does not violate any bound, although the pure level portion
   ## violates the Bose/Bush bound
   ## for almost pure level: also check pure level portions!

oa_feasible(24, c(2,4,3,4))
   ## violates divisibility by the LCM of all products of pairs
oa_feasible(48, c(2,4,3,4,2))
   ## TRUE and indeed feasible
```

---

`param.design` *Function to generate Taguchi style parameter designs*

---

## Description

The functions create parameter designs for robustness experiments and signal-to-noise investigations with inner and outer arrays and facilitate their formatting and data aggregation.

## Usage

```
param.design(inner, outer, direction="long", responses=NULL, ...)
paramtowide(design, constant=NULL, ...)
```

## Arguments

| | |
|---|---|
| inner | an experimental design for the inner array, data frame of class design; as function param.design does not randomize, its runs should already be randomized |
| outer | an experimental design for the outer array, data frame of class design or vector |
| direction | character taking the values "wide" or "long"; if long, the outer array runs for each inner array run are listed underneath each other; if wide, they are listed within the same row |
| responses | NULL, or character vector of response names;<br>for the long format, there are no response columns if responses is NULL, while response columns of the specified name(s) containing NA values are generated if responses is specified;<br>for the wide format, response columns are always generated (one column per run of the outer array for each response): if responses is NULL, response columns are called "y.1", "y.2" etc., if responses is specified, a set of response columns for each specified name is generated |
| design | parameter design in long format (created by function param.design) |
| constant | character vector giving names of variables in addition to the experimental factors of the inner array that are constant over outer array runs for each inner array run |
| ... | currently not used |

## Details

A parameter design is an experimental plan for setting the so-called "control parameters" such that they achieve the intended function and at the same time minimize the effects of the so-called "noise parameters". Note that the word parameters is used here in an engineering sense rather than in the typical sense it is used in statistics. The experiment crosses the control factors in the "inner array" with the noise factors in the "outer array".

Function param.design uses function cross.design for creating an inner/outer array crossed design. There will be data aggregation functions for such designs in the near future.

Note that designs created by param.design are not properly randomized, as they are conducted in the Taguchi inner / outer array sense with the runs of the inner array as whole plots and the factors of the outer array as split-plot factors. With analysis methods that work on data aggregated over the outer array this is appropriate. If analysis of control and noise factor designs is to be conducted in a combined approach, the experiment should be fully randomized. This can be done using function cross.design directly (cf. example there).

## Value

A data frame of class design with type "param" or "FrF2.param" for long version inner/outer array designs, and type of the inner array suffixed with ".paramwide" for wide version inner/outer array

designs. The design.info attribute of such designs has the following extraordinary elements:

In long format, there are the same elements as for type crossed from function cross.design, and the additional elements inner and outer that give the names of the inner and outer array variables.

In wide format, the design.info information refers to the inner array, the elements cross... something are no longer available (except for cross.types), and the element outer contains the outer array design the rows of which correspond to the response columns. The additional element format with value "innerouterWide" indicates the wide format (introduced for analogy to wide repeated measures designs), and responselist shows the responses and their respective columns in support of subsequent aggregation. Finally, if there are variables that are neither experimental factors nor responses and change within one run of the inner array, these are listed in restlist.

## Note

This function is still experimental.

## Author(s)

Ulrike Groemping

## References

NIST/SEMATECH e-Handbook of Statistical Methods, Section 5.5.6 (What are Taguchi Designs?), accessed August 11, 2009. https://www.itl.nist.gov/div898/handbook/pri/section5/pri56.htm

## See Also

See Also cross.design

## Examples

```
## It is recommended to use param.design particularly with FrF2 designs.
## For the examples to run without package FrF2 loaded,
## oa.design designs are used here.

## quick preliminary checks to try out possibilities
control <- oa.design(L18, columns=1:4, factor.names=paste("C",1:4,sep=""))
noise <- oa.design(L4.2.3, columns=1:3, factor.names=paste("N",1:3,sep=""))
## long
long <- param.design(control,noise)
## wide
wide <- param.design(control,noise,direction="wide")
wide
long

## use proper labelled factors
## should of course be as meaningful as possible for your data
fnc <- c(list(c("current","new")),rep(list(c("type1", "type2","type3")),3))
names(fnc) <- paste("C", 1:4, sep="")
```

```
control <- oa.design(L18, factor.names=fnc)
fnn <- rep(list(c("low","high")),3)
names(fnn) <- paste("N",1:3,sep="")
noise <- oa.design(L4.2.3, factor.names = fnn)
ex.inner.outer <- param.design(control,noise,direction="wide",responses=c("force","yield"))
ex.inner.outer
## export e.g. to Excel or other program with which editing is more convenient
## Not run:
   ### design written to default path as html and rda by export.design
   ### html can be opened with Excel
   ### data can be typed in
   ### for preparation of loading back into R,
   ###     remove all legend-like comment that does not belong to the data table itself
   ###     and store as csv
   ### reimport into R using add.response
   ### (InDec and OutDec are for working with German settings csv
   ###     in an R with standard OutDec, i.e. wrong default option)
   getwd()  ## look at default path, works on most systems
   export.design(ex.inner.outer, OutDec=",")
   add.response("ex.inner.outer", "ex.inner.outer.csv", "ex.inner.outer.rda", InDec=",")

## End(Not run)
```

---

`planor2design` *Convert matrix, data frame or object of class planordesign to object of class design*

---

## Description

function to convert matrix, data frame or object of class planordesign to class design (allowing use of convenience functions, particularly plotting with mosaic plots)

## Usage

```
data2design(x, quantitative = rep(FALSE, ncol(x)), ...)
planor2design(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | an object of class `data.frame`, `matrix` (function `data2design`) or `planordesign` |
| `quantitative` | a logical vector, indicating which factors are quantitative;<br>defaults to all factors being qualitative |
| `...` | currently not used |

### Details

For matrices and data frames, an unreplicated and unrandomized design is assumed (not crucial, but the some entries of the `design.info` attribute and the entire `run.order` attribute of the result will be wrong otherwise). Per default, all factors are treated as qualitative and thus made into factors, if they are not factors already.

Items of the S4 class `planordesign` are regular factorial designs created by package **planor** (the designs itself is in the slot `design`). Function `planor2design` transforms them into objects of the S3 class `design`; currently, only the most basic information is included (nunit and the factor information); the design is assumed to be unrandomized and unreplicated.

### Value

an object of class `design` with the `type` and `creator` element of `design.info` given as `external` or `planor`. For designs of type `planor`, the `generators` element of the `design.info` attribute contains the `designkey` from the original `planor` design.

### Author(s)

Ulrike Groemping

### See Also

See also: the `planordesign` class of package **planor** (if that package is available), `design`, `plot.design`

---

`Plotting class design objects`
*Plotting class design objects*

---

### Description

The plot method for class design objects; other methods are part of a separate help page.

### Usage

```
## S3 method for class 'design'
plot(x, y=NULL, select=NULL, selprop=0.25, ask=NULL, ...)
```

### Arguments

| | |
|---|---|
| `x` | data frame of S3 class `design` |
| `y` | a character vector of names of numeric variables in `x` to be plotted as responses, or a numeric response vector, or a numeric matrix containing response columns, or a data frame of numeric response variables (the latter would not work when directly using function `plot.design` from package graphics) |

`select` Specification of selected factors through option `select` has been introduced in order to obtain manageable plot sizes. For example, mosaic plots are most easily readible for up to three or at most four factors. Main effects plots with too many factors may also be hard to read because of overlapping labeling. `select` can also be used for bringing the factors into a desirable order.
`select` can be
a vector of integers with position numbers of experimental factors,
a character vector of factor letters,
or a character vector of factor names for factors to be selected for plotting; in case of ambiguity, factor names take precedence over factor letters.
The following choices are of interest for creating mosaic plots of the design factors; any response data will be ignored.
`select` can be
a list of numeric vectors (all of equal length) specifying the tuples to be plotted (a length one list (insted of the numeric vector itself) allows to plot the design table as a mosaic plot instead of showing a main effects plot, if the design has responses)
one of the special character strings `"all2"`, `"all3"` or `"all4"` for obtaining mosaic plots of all pairs, triples or quadruples of (a selection of) factors (see Details section),
a list with a numeric vector with position numbers of experimental factors as the first and one of the above special character strings as the second element for requesting all tuples of a subset of the factors,
or a list with a single factor position number as the first and one of the above special character strings as the second element for requesting all tuples that include the specified single factor,
or any of the character strings `"complete"`, `"worst"`, `"worst.rel"`, `"worst.parft"` or `"worst.parftdf"` for automatic selection of the projections with the worst confounding to be plotted (see Details section)

`selprop` a number between 0 and 1 indicating which proportion of worst cases to plot in case `select=worst` or `select=worst.rel` is to be plotted (see Details section). The default is useful for small designs only. For large designs, reduce this number !

`ask` a logical;
default behavior if `ask=NULL`: `ask` is set to `TRUE` if multiple plots are requested and the current graphics device is interactive (or none is open but be the next to be opened device is interactive) and `FALSE` otherwise

`...` further arguments to functions `plot`, `mosaic`, or the function `plot.design` from package graphics;
For experts, option `sub` with the special settings `"GR"`, `"A"`,`"rA"`, `"sumPARFT"` or `"sumPARFTdf"` can be used to create sub titles that display the generalized resolution, absolute or relative word lengths (see `generalized.word.length`). All other specifications for `sub` should work as expected.

## Details

Items of class `design` are data frames with attributes, that have been created for conducting experiments. Apart from the plot method documented here, separate files document the methods

formula.design, lm.design, and further methods.

The method for plot calls the method available in package graphics (see plot.design) wherever this makes sense (x not of class design, x of class design but not following the class design structure defined in package **DoE.base**, and x a design with all factors being R-factors and at least one response available).

Function plot.design from package **graphics** is not an adequate choice for designs without responses or designs with experimental factors that are not R-factors.
For designs with all factors being R-factors and no response defined (e.g. a freshly-created design from function oa.design), function plot.design creates a mosaic plot of the frequency table of the design, which may be quite useful to understand the structure for designs with relatively few factors (cf. example below; function plot.design calls function mosaic for this purpose). It will generally be necessary to specify the select argument, if the design is not very small. If select is not specified although there are more than four factors, select=1:4 is chosen as the default.
For designs with at least one experimental factor that is not an R-factor, function plot.design calls function plot.data.frame in order to create a scatter plot matrix.
Currently, there is no good method for plotting designs with mixed qualitative and quantitative factors.

If option select is set to "all2", "all3" or "all4", or a list with a numeric vector as its first element and one of these as the second element, or with select as any of "complete", "worst", "worst.rel", "worst.parft" or "worst.parftdf", response variables are ignored, and mosaic plots are created.
These requests usually ask for several plots; note that the plots are created one after the other; with an interactive graphics device, the default is that they overwrite each other after a user confirmation for the next plot, which allows users to visually inspect them one at a time; under Windows, the plotting series can be aborted using the Esc-key. With non-interactive graphics devices, the default is ask=FALSE (e.g. for storing all the plots in a multi-page file, see examples).

If option select is any of "all2", "all3" or "all4", mosaic plots of all pairs, triples or quadruples of factors are created as specified.
Note that "all2" is interesting for non-orthogonal designs only, e.g. ones created by function Dopt.design.

If option select is set to "complete", "worst" "worst.rel", "worst.parft" or "worst.parftdf", the worst case tuples to be displayed are selected by function tupleSel.

## Value

The plot method is called for its side effects and returns NULL.

## Note

The package is currently subject to intensive development; most key functionality is now included. Some changes to input and output structures may still occur.

## Author(s)

Ulrike Groemping

## References

Groemping, U (2014) Mosaic plots are useful for visualizing low order projections of factorial designs. To appear in *The American Statistician* https://www.tandfonline.com/action/showAxaArticles?journalCode=utas20.

## See Also

See also the following functions known to produce objects of class design: FrF2, pb, fac.design, oa.design, and function plot.design from package graphics; a method for function lm is described in the separate help file lm.design.

## Examples

```
#### Examples for plotting designs
oa12 <- oa.design(nlevels=c(2,2,6))
  ## plotting a design without response (uses function mosaic from package vcd)
  plot(oa12)
  ## equivalent to mosaic(~A+B+C, oa12)
  ## alternative order: mosaic(~C+A+B, oa12)
  plot(oa12, select=c(3,1,2))
  ## using the select function: the plots show that the projection for factors
  ## C, D and E (columns 3, 14 and 15 of the array) is a full factorial,
  ## while A, D and E (columns 1, 14, and 15 of the array) do not occur in
  ## all combinations
  plan <- oa.design(L24.2.13.3.1.4.1,nlevels=c(2,2,2,3,4))
  plot(plan, select=c("E","D","A"))
  plot(plan, select=c("E","D","C"))
  ## Not run:
  plot(plan, select="all3")
  plot(plan, select=list(c(1,3,4,5), "all3"))
  ## use the specialist version of option sub
  plot(plan, select=list(c(1,3,4,5), "all3"), sub="A")
  ## create a file with mosaic plots of all 3-factor projections
  pdf(file="exampleplots.pdf")
  plot(plan, select="all3", main="Design from L24.2.13.3.1.4.1 in default column order)")
  plot(plan, select="worst", selprop=0.3, sub="A")
  dev.off()
  ## the file exampleplots.pdf is now available within the current working
  ## directory

## End(Not run)

  ## plotting a design with response
  y=rnorm(12)
  plot(oa12, y)
  ## plot design with a response included
  oa12.r <- add.response(oa12,y)
  plot(oa12.r)
  ## plotting a numeric design (with or without response,
  ##   does not make statistical sense here, for demo only)
  noa12 <- qua.design(oa12, quantitative="all")
  plot(noa12, y, main="Scatter Plot Matrix")
```

print.oa *Function to Print oa Objects with a Lot of Added Info*

## Description

The function suppresses printing of voluminous info attached as attributes to oa objects.

## Usage

```
## S3 method for class 'oa'
print(x, ...)
```

## Arguments

| | |
|---|---|
| x | the oa object to be printed |
| ... | further arguments for default print function |

## Details

The function currently removes all attributes except `origin`, `class`, `dim`, `dimnames` before printing. If available, status information from the `MIPinfo` attribute is printed.
Additionally, the names of unusual attributes are printed. They can also be printed separately by running `names(attributes(x))`; to access an attribute, run `attr(x, "MIPinfo")`, for example.

## Value

The function is used for its side effects and does not return anything.

## Author(s)

Ulrike Groemping

## See Also

See also `print.default` and `str`

qua.design *Function to switch between qualitative and quantitative factors and different contrast settings*

## Description

The function allows to switch between qualitative and quantitative factors and different contrast settings.

## Usage

```
qua.design(design, quantitative = NA, contrasts = character(0), ...)
change.contr(design, contrasts=contr.treatment)
```

## Arguments

| | |
|---|---|
| design | an experimental design, data frame of class design |
| quantitative | can be<br>EITHER<br>one of the single entries<br>NA for setting all factors to the default coding for class design (cf. details),<br>"all" for making all factors quantitative (=numeric),<br>"none" for making all factors qualitative (=factor)<br><br>OR<br>an unnamed vector of length nfactors with an entry TRUE, NA or FALSE for each factor, where TRUE makes a factor into a numeric variable, and FALSE makes it into a factor with treatment contrasts, and NA reinstates the default factor settings;<br><br>OR<br>a named vector (names from the factor names of the design) with an entry TRUE, NA or FALSE for each named factor (implying no change for the omitted factors) |
| contrasts | only takes effect for factors for which quantitative is FALSE;<br>the default character(0) does not change any contrasts vs.~the previous or default contrasts.<br><br>For customizing, a<br>character string<br>OR a character vector with a contrast name entry for each factor OR a named character vector of arbitrary length from 1 to number of factors<br><br>can be given; the names must correspond to names of factors to be modified, and entries must be names of contrast functions. The contrast functions are then applied to the respective factors with the correct number of levels. |

Possible contrast function names include (at least) contr.FrF2 (for number of levels a power of 2 only), contr.helmert, contr.treatment, contr.SAS, contr.sum, contr.poly. CAUTION: Function qua.design checks whether the contrast names actually define a function, but it is not checked whether this function is a valid contrast function.

... currently not used

## Details

With function qua.design, option quantitative has the following implications:
An experimental factor for which quantitative is TRUE is recoded into a numeric variable.
An experimental factor for which quantitative is NA is recoded into an R-factor with the default contrasts given below.
An experimental factor for which quantitative is FALSE is recoded into an R-factor with treatment contrasts (default) or with custom contrasts as indicated by the contrasts parameter.
If the intention is to change contrasts only, function change.contr is a convenience interface to function qua.design.

The default contrasts for factors in class design objects (exception: purely quantitative design types like lhs or rsm designs) depend on the number and content of levels:
2-level experimental factors are coded as R-factors with -1/1 contrasts,
experimental factors with more than two quantitative (=can be coerced to numeric) levels are coded as R factors with polynomial contrasts (with scores the numerical levels of the factor),
and qualitatitve experimental factors with more than two levels are coded as R factors with treatment contrasts.

Note that, for 2-level factors, the default contrasts from function qua.design differ from the default contrasts with which the factors were generated in case of functions fac.design or oa.design. Thus, for recreating the original state, it may be necessary to explicity specify the desired contrasts.

Function change.contr makes all factors qualitative. Per default, treatment contrasts (cf. contr.treatment) are assigned to all factors. The default contrasts can of course be modified.

Warning: It is possible to misuse these functions especially for designs that have been combined from several designs. For example, while setting factors in an lhs design (cf. lhs.design) to qualitative is prevented, if the lhs design has been crossed with another design of a different type, it would be possible to make such a nonsensical modification.

## Value

A data frame of class design; the element quantitative of attribute design.info, the data frame itself and the desnum attribute are modified as appropriate.

## Author(s)

Ulrike Groemping

## Examples

```
## usage with all factors treated alike
y <- rnorm(12)
plan <- oa.design(nlevels=c(2,6,2))
```

```
lm(y~.,plan)
lm(y~., change.contr(plan))   ## with treatment contrasts instead
plan <- qua.design(plan, quantitative = "none")
lm(y~.,plan)
plan <- qua.design(plan, quantitative = "none", contrasts=c(B="contr.treatment"))
lm(y~.,plan)
plan <- qua.design(plan, quantitative = "none")
lm(y~.,plan)

plan <- qua.design(plan, quantitative = "all")
lm(y~.,plan)
plan <- qua.design(plan)  ## NA resets to default state
lm(y~.,plan)

## usage with individual factors treated differently
plan <- oa.design(factor.names = list(liquid=c("type1","type2"),
     dose=c(0,10,50,100,200,500), temperature=c(10,15)))
str(undesign(plan))
## Not run:
## would cause an error, since liquid is character and cannot be reasonably coerced to numeric
plan <- qua.design(plan, quantitative = "all")

## End(Not run)
plan <- qua.design(plan, quantitative = "none")
str(undesign(plan))

plan <- qua.design(plan, quantitative = c(dose=TRUE,temperature=TRUE))
str(undesign(plan))
## reset all factors to default
plan <- qua.design(plan, quantitative = NA)
str(undesign(plan))
desnum(plan)
## add a response
y <- rnorm(12)
plan <- add.response(plan,y)
## set dose to treatment contrasts
plan <- qua.design(plan, quantitative = c(dose=FALSE), contrasts=c(dose="contr.treatment"))
str(undesign(plan))
desnum(plan)
```

---

Reshape designs with repeated measurements

*Reshape designs with repeated measurements*

---

## Description

Convenience functions to reshape a design with repeated measurements from long to wide or vice versa

## Usage

```
### generic function
reptowide(design, constant=NULL, ...)
reptolong(design)
```

## Arguments

| | |
|---|---|
| `design` | a data frame of S3 class `design`.<br>For function `reptowide`, the design must have repeated measurements (`repeat.only=TRUE` in `design.info` attribute).<br>For `reptolong`, the design must be in the wide form produced by function `reptowide`. |
| `constant` | `NULL` or character vector; if `design` contains variables other than the experimental factors and the block column (e.g. covariables) that do not change over repeated measurements within the same experimental unit, `constant` must be a character vector with the respective variable names |
| `...` | currently not used |

## Details

Both functions leave the design unchanged (with a warning) for all class `design` objects that are not of the required repeated measurements form.

If `design` is not of class `design`, an error is thrown.

The `reptowide` function makes use of the function `reshape` in package **stats**, the `reptolong` function does not.

## Value

A data frame of class `design` with the required reshaping.

The `reptowide` function returns a design with one row containing all the repeated measurements for the same experimental setup (therefore wide), the `reptolong` function reshapes a wide design back into the long form with all repeated measurements directly underneath each other.

The attributes of the design are treated along with the data frame itself: The `reptowide` function resets elements of the `design.info` attribute (`response.names`, `repeat.only`) and adds the new elements `format` with value "repeatedMeasuresWide", `responselist` and, if there are variables that are neither experimental factors nor responses, `restlist` for those of these that do change with repeated measurements. The `reptolong` function reinstates the original long version.

Note that the order of variables may change, if there are any variables in addition to the factors and responses.

## Note

The package is currently subject to intensive development; most key functionality is now included. Some changes to input and output structures may still occur.

## Author(s)

Ulrike Groemping

## See Also

See Also FrF2, pb, fac.design, oa.design

## Examples

```
### design without response data
### response variable y is added per default
plan <- oa.design(nlevels=c(2,6,2), replication=2, repeat.only=TRUE)
pw <- reptowide(plan)  ## make wide
pl <- reptolong(pw)  ## make long again

### design with response and further data
y <- rexp(24)
temp <- rep(sample(19:30),each=2)  ## constant covariable
prot.id <- factor(Letters[1:24]) ## non-constant character covariable
plan.2 <- add.response(plan, y)
plan.2$temp <- temp     ## not response
plan.2$prot.id <- prot.id   ##not response
plan.2
reptowide(plan.2, constant="temp")
```

---

# show.oas *Function to display list of available orthogonal arrays*

---

## Description

This function allows to inspect the list of available orthogonal arrays, optionally specifying selection criteria

## Usage

```
show.oas(name = "all", nruns = "all", nlevels = "all", factors = "all",
    regular = "all", GRgt3 = c("all", "tot", "ind"), Rgt3 = FALSE, show = 10,
    parents.only = FALSE, showGRs = FALSE, showmetrics = FALSE, digits = 3)
```

## Arguments

| | |
|---|---|
| name | character string or vector of character strings giving name(s) of (an) orthogonal array(s); results in an error if name does not contain any valid name; warns if name contains any invalid name |
| nruns | the requested number of runs or a 2-element vector with a minimum and maximum for the number of runs |

| | |
|---|---|
| nlevels | a vector of requested numbers of levels for a set of factors in question, must contain integers > 1 only;<br>nlevels cannot be specified together with factors |
| factors | a list with the two elements nlevels and number, which are both integer vectors of equal length;<br>nlevels contains the number of levels and number the number of factors for the corresponding number of levels |
| regular | either unrestricted (the default "all"), a logical which requests (TRUE) or rejects (FALSE) regular arrays, or the character string "strict" to request strictly regular arrays, for which all confounded factors are *completely* confounded with a 2-factor interaction of two other factors (the latter are fixed level arrays or crossed arrays) |
| GRgt3 | either unrestricted (the default "all"), or a character string which requests GR ("tot") or GRind ("ind") to be larger than 3 |
| Rgt3 | logical requesting inclusion of standard resolution 3 arrays as listed in oacat per default, and restricting the output to arrays of resolution at least IV (as listed in oacat3), if changed to TRUE |
| show | an integer number specifying how many arrays are to be listed (upper bound), or the character string "all" for showing all arrays, no matter how many. The default is to show 10 arrays. show = 0 switches off the display of the result and only returns a value. Since August 2018, the number refers to stronger and weaker arrays, separately. |
| parents.only | logical specifying whether to show only parent arrays or child arrays as well; the default is FALSE for inclusion of child arrays |
| showGRs | logical specifying whether to show the generalized resolution quality metrics with the resulting arrays; the default is FALSE. If set to TRUE, three metrics are displayed (see Details section). |
| showmetrics | logical specifying whether to show all array quality metrics with the resulting arrays; the default is FALSE. If set to TRUE, several metrics are displayed (see Details section). |
| digits | integer number of significant digits to show for GR and A metrics; irrelevant, if showmetrics is FALSE |

## Details

The function shows the arrays that are listed in the data frames oacat or oacat3.

For child arrays that have to be generated with a lineage rule (can be automatically done with function oa.design), the lineage is displayed together with the array name. The option parent.only = TRUE suppresses printing and output of child arrays. The structure of the lineage entry is documented under oacat.

If display of metrics is requested with showmetrics=TRUE, the printed output shows the metrics GR*, GRind*, regular (logical, whether regular or not), SCones* (number of squared canonical correlations that are 1), and the numbers of words of lengths 3 to 8 (A3 to A8). showGRs=TRUE requests the metrics marked with asterisks only (without SCones in case GRgt3="ind"). More information on all these metrics can be found here

## Value

A data frame with the three columns `name`, `nruns` and `lineage`, containing the array name, the number of runs and - if applicable - the lineage for generating the array from other arrays. The lineage entry is empty for parent arrays that are either directly available in the package and can be accessed by giving their name (e.g. `L18.3.6.6.1`) or are full factorials (e.g. `L28.4.1.7.1`). If further information has been requested (e.g. with `showmetrics=TRUE`), the data frame contains additional columns.

If no array has been found, the returned value is `NULL`.

## Note

Thanks to Peter Theodor Wilrich for proposing such a function.

## Author(s)

Ulrike Groemping

## References

Kuhfeld, W. (2009). Orthogonal arrays. Website courtesy of SAS Institute https://support.sas.com/techsup/technote/ts723b.pdf and references therein.

Mee, R. (2009). *A Comprehensive Guide to Factorial Two-Level Experimentation*. New York: Springer.

## See Also

`oa.design` for using the arrays from `oacat` in design creation
`oacat` for the data frames underlying the function

## Examples

```
 ## the first 10 orthogonal arrays with 24 to 28 runs
 show.oas(nruns = c(24,28))
 ## the first 10 orthogonal arrays with 24 to 28 runs
 ## excluding child arrays
 show.oas(nruns = c(24,28), parents.only=TRUE)
## the orthogonal arrays with 4 2-level factors, one 4-level factor and one 5-level factor
 show.oas(factors = list(nlevels=c(2,4,5),number=c(4,1,1)))
 ## show them all with quality metrics
 show.oas(factors = list(nlevels=c(2,4,5),number=c(4,1,1)), show=Inf, showmetrics=TRUE)
 ## pick only those with no complete confounding of any degrees of freedom
show.oas(factors = list(nlevels=c(2,4,5),number=c(4,1,1)), GRgt3="ind", showmetrics=TRUE)
## the orthogonal arrays with 4 2-level factors, one 7-level factor and one 5-level factor
 show.oas(factors = list(nlevels=c(2,7,5),number=c(4,1,1)))
 ## the latter orthogonal arrays with the nlevels notation
 ## (that can also be used in a call to oa.design subsequently)
 show.oas(nlevels = c(2,7,2,2,5,2))
 ## calling arrays by name
 show.oas(name=c("L12.2.11", "L18.2.1.3.7"))
```

SN *Function for the signal-to-noise ratio 10 * log10(mean^2/var)*

## Description

Function for the signal-to-noise ratio 10 * log10(mean^2/var)

## Usage

```
SN(x)
```

## Arguments

| | |
|---|---|
| x | a data vector to take the S/N ratio over |

## Details

Taguchi proposes three different versions of S/N-ratio. In line with Box, Hunter and Hunter (2005), only the one for target-optimization is given here, as it is invariant against linear transformation.

## Value

a number (10 * log10(mean^2/var))

## Note

This package is currently under intensive development. Substantial changes are to be expected in the near future.

## Author(s)

Ulrike Groemping

## References

Box G. E. P, Hunter, W. C. and Hunter, J. S. (2005) *Statistics for Experimenters, 2nd edition*. New York: Wiley.

## See Also

See also `aggregate.design`; function `SN` has been developed for use with aggregating parameter designs

## Examples

```
x <- rexp(10)
SN(x)
10 * log10(mean(x)^2/var(x))
20 * log10(mean(x)/sd(x))
```

VSGFS *VSGFS: an experiment using an optimized orthogonal array in 72 runs*

## Description

VSGFS: an experiment using an optimized orthogonal array in 72 runs

## Usage

```
VSGFS
```

## Format

VSGFS is a data frame of class `design` with seven experimental factors and three response variables. The data have been published in Vasilev et al. (2014).

The experimental factors, all stored as R factors, with their levels are

| | | |
|---|---|---|
| [,1] | Light | Lght-, Lght+ |
| [,2] | ShakFreq | SF-, SF+ |
| [,3] | InocSize | IS-, IS+ |
| [,4] | FilledVol | FV-, FV0, FV+ |
| [,5] | CM | CM-, CM+ |
| [,6] | Carbo | Suc, Gluc, Mannit (Sucrose, Glucose, Mannitol) |
| [,7] | Cyclodextrin | CD1, CD2, CD3, CD4 (beta, methyl-beta, triacetyl-beta, none) |

The response variables, all stored as numerical variables, are

| | | |
|---|---|---|
| [,8] | Biomass | fresh weight in g |
| [,9] | Content | geraniol content in $\mu$g per g fresh weight |
| [,10] | Yield | geraniol yield in $\mu$g per flask |

## Details

The data set comes from an experiment that was created with function `oa.design` using the array `L72.2.43.3.8.4.1.6.1`. Column selection within the array was done with option `columns="min34"` that picks the first set of columns obtained by function `oa.min34`. (Optimization takes quite a while, so that the design was reconstructed later by explicitly requesting the optimum set of columns.)

Design creation and the experiment itself were conducted at the Fraunhofer IME in Aachen by Nikolay Vasilev and colleagues. More detail on the experiment and the variables can be found in Vasilev et al. (2014).

The design was created under an R version before 3.6.0. For reproducing its creation under R 3.6.0 and later, it is therefore necessary to switch to the previous version of random number generation (using the `RNGkind` function, see examples section). Note that the previous discrete random uniform random number generator was not perfectly uniform, especially for very large samples; for randomizing experiments of typical sizes (like this one), this problem can be neglected.

## Author(s)

Ulrike Groemping

## References

Vasilev, N., Schmidt, C., Groemping, U., Fischer, R. and Schillberg, S. (2014). Assessment of Cultivation Factors that Affect Biomass and Geraniol Production in Transgenic Tobacco Cell Suspension Cultures. *PLoS ONE* **9**(8): e104620. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0104620.

## See Also

See also oacat, show.oas, oa.min34, oa.design

## Examples

```
## code used for creating the data frame
## option levordold is needed, because the level ordering
## changed (improved) with version 0.27
## and the design was originally created with an earlier version
## Not run:
  if (getRversion()>='3.6.0') RNGkind(sample.kind="Rounding")
  VSGFS <- oa.design(ID=L72.2.43.3.8.4.1.6.1,
   nlevels=c(2,2,2,3,2,3,4),
   columns=c(4,22,37,46,41,48,52),
   factor.names=list(Light=c("Lght-","Lght+"),
      ShakFreq=c("SF-","SF+"),
      InocSize=c("IS-","IS+"),
      FilledVol=c("FV-","FV0", "FV+"),
      CM=c("CM-","CM+"),
      Sugar=c("Suc", "Gluc", "Mannit"),
      CDs=c("CD1","CD2","CD3","CD4")),
   seed = 9, randomize=TRUE, levordold=TRUE)
  if (getRversion()>='3.6.0') RNGkind(sample.kind="default")

response <- as.data.frame(scan(what=list(Biomass=0, Content=0, Yield=0), sep=" "))
5.80 24.13 139.98
4.97 16.96 84.28
1.28 21.08 26.99
6.83 17.71 120.95
0.86 21.28 18.30
4.09 18.86 77.14
2.39 17.08 40.81
4.05 17.84 72.23
5.84 17.74 103.61
3.38 18.08 61.11
0.40 24.82 9.93
3.86 18.10 69.88
4.58 21.29 97.49
6.29 17.32 108.91
4.85 15.50 75.17
1.25 23.14 28.92
```

```
2.09 18.43 38.51
4.26 17.75 75.62
4.78 18.53 88.57
6.63 17.82 118.14
0.77 18.79 14.47
4.89 18.23 89.15
4.53 17.69 80.11
4.27 18.05 77.07
3.90 15.84 61.77
4.15 18.73 77.74
3.95 17.12 67.63
6.92 16.86 116.68
5.00 16.96 84.80
0.37 21.79 8.06
2.36 19.57 46.18
5.11 18.13 92.66
4.69 17.38 81.50
1.20 19.57 23.49
1.76 17.98 31.65
6.21 17.03 105.76
5.63 15.71 88.43
3.98 18.42 73.32
2.31 19.38 44.76
1.86 18.41 34.25
4.22 17.93 75.68
2.77 17.17 47.55
0.40 23.10 9.24
1.42 18.89 26.83
1.54 17.44 26.86
5.03 17.40 87.53
8.70 14.41 125.38
3.21 19.29 61.92
5.36 18.46 98.93
3.87 16.89 65.35
7.70 18.60 143.20
1.71 17.67 30.22
4.38 16.79 73.54
2.24 19.61 43.92
3.79 19.35 73.35
3.09 18.67 57.70
1.57 17.64 27.70
5.43 18.45 100.19
3.86 17.09 65.96
7.44 19.07 141.85
5.87 17.13 100.53
2.65 17.51 46.39
6.14 15.85 97.34
6.32 14.80 93.56
5.19 16.53 85.78
5.09 17.30 88.04
4.40 17.52 77.08
1.68 21.89 36.78
0.93 23.06 21.45
```

```
1.79 22.88 40.95
2.64 18.38 48.52
7.78 16.22 126.19


VSGFS <- add.response(VSGFS, response)
VSGFS$Sugar <- relevel(VSGFS$Sugar, "Suc")
VSGFS$FilledVol <- relevel(VSGFS$FilledVol, "FV0")
VSGFS$FilledVol <- relevel(VSGFS$FilledVol, "FV-")

## End(Not run)
```

# Index