

Package ‘DoE.base’

January 2, 2012

Title Full factorials, orthogonal arrays and base utilities for DoE packages

Version 0.22-8

Depends R(>= 2.10.0), stats, relimp, utils, graphics, tcltk, vcd,conf.design

Date 2011-11-15

Author Ulrike Groemping, with contributions by Boyko Amarov

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

Description This package creates full factorial experimental designs and designs based on orthogonal arrays for (industrial) experiments. Additionally, it provides some utility functions used also by other DoE packages.

License GPL (>= 2)

LazyLoad yes

LazyData yes

Encoding latin1

URL <http://prof.beuth-hochschule.de/groemping/DoE/>,
<http://prof.beuth-hochschule.de/groemping/>

Repository CRAN

Date/Publication 2011-11-15 13:10:09

R topics documented:

DoE.base-package	2
add.response	3
block.catlg3	5
Class design and accessors	6
contr.FrF2	11
cross.design	12
export.design	14

fac.design	17
factorize	21
fix.design	23
formula.design	24
genChild	26
generalized.word.length	28
iscube	34
lm and aov method for class design objects	35
Methods for class design objects	40
oa.design	47
oacat	53
param.design	55
qua.design	57
Reshape designs with repeated measurements	60
show.oas	62
SN	64

Index	66
--------------	-----------

DoE.base-package	<i>Full factorials, orthogonal arrays and base utilities for DoE packages</i>
------------------	---

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 an early development phase and not fully mature. 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); so far, only the parent arrays have been implemented, but many more arrays will follow soon. 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!

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.

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 <http://support.sas.com/techsup/technote/ts723.html>.

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 <code>design</code> object, to which responses are to be added
response	EITHER a numeric vector, numeric matrix or data frame with at least one numeric variable (the treatment of these is explained in the details section) OR 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 <code>InDec</code> 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 <code>OutDec</code> option (viewable under <code>options("OutDec")</code>), and also governs whether the csv-file is read with <code>read.csv</code> or with <code>read.csv2</code> : separator semicolon goes with decimal comma and triggers use of <code>read.csv2</code> , separator comma goes with decimal point and triggers use of <code>read.csv</code> .)
tol	tolerance for comparing numerical values; useful for designs with numeric factors and for partial replacement of response values; the value is used in comparisons of design and response via <code>all.equal</code> ; errors from peculiar rounding behavior of spreadsheet programs can be prevented by allowing a larger <code>tol</code>
...	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^p 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

```

design
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) <- 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 <code>undesign</code>)
value	<p>an appropriate replacement value:</p> <p>a numeric version of the design matrix for function <code>desnum</code> (usage not encouraged for non-experts!)</p> <p>a run order data frame for function <code>run.order</code> (usage not encouraged for non-experts!)</p> <p>a list with appropriate design information for function <code>design.info</code> (usage not encouraged for non-experts!)</p> <p>for function <code>'factor.names<-'</code> a character vector of new factor names (levels remain unchanged) or a named list of level combinations for the factors, like <code>factor.names</code> in function fac.design</p> <p>for function <code>'response.names<-'</code> a character vector of response names referring to variables which are already available in design</p>
contr.modify	<p>logical to indicate whether contrasts are to be modified to match the new levels; relevant for R factors only, not for numeric design variables;</p> <p>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; if FALSE, the contrasts remain unchanged from their previous state.</p> <p>If solely the contrasts are to be changed, function change.contr is preferable.</p>
remove	<p>logical to indicate whether responses not indicated in <code>value</code> are to be removed from the design altogether.</p> <p>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.</p>

<code>colnames</code>	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 <code>desnum</code> attribute of the design may have to be manually modified for removing the respective columns in some cases.
<code>matrix</code>	matrix, data frame or also object of class design that is to be ordered column by column
<code>decreasing</code>	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 or [fix.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 the integer seed handed to the function call by the user

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

creator contains the call or (in the future) the list of menu settings within the currently developed package **RcmdrPlugin.DoE** that led to creation of the design.

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

<code>desnum</code>	returns a numeric matrix, the corresponding replacement function modifies a class design object
<code>run.order</code>	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
<code>design.info</code>	returns the <code>design.info</code> attribute of the design; the corresponding replacement function modifies a class design object
<code>factor.names</code>	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
<code>'factor.names<-'</code>	returns a class design object with modified factor names information (renamed factors and/or changed factor levels);
<code>response.names</code>	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
<code>'response.names<-'</code>	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.
<code>col.remove</code>	returns a class design object with some columns removed from both the design itself and the <code>desnum</code> attribute. Response columns may be removed, but factor or block columns may not.
<code>ord</code>	returns an index vector that orders the matrix or data frame; for example, <code>design[ord(design),]</code> orders the design in increasing order with respect to the first, then the second etc. factor.

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 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

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)
```

Arguments

`n` power of 2; number of levels of the factor for which contrasts are to be generated

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)
```

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) 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; can also be a vector if ... is not used; 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	seed for random number generation

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 accommodate 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 accommodated; 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:

<code>cross.nruns</code>	vector of run numbers of individual designs
<code>cross.nfactors</code>	vector of numbers of factors of individual designs
<code>cross.types</code>	vector of types of individual designs
<code>cross.randomize</code>	vector of logicals (randomized or not) of individual designs
<code>cross.seed</code>	vector of seeds of individual designs
<code>cross.replications</code>	vector of numbers of replications of individual designs
<code>cross.repeat.only</code>	vector of logicals (repeat.only or not) of individual designs
<code>cross.map</code>	list with the map vectors for component designs of type <code>FrF2.estimable</code>
<code>cross.selected.columns</code>	NULL (if no oa type design) or list of column vectors for each design
<code>cross.nlevels</code>	list with the <code>nlevels</code> 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.

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)
```

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")$OutDec, replace = FALSE, ...)
html(object, ...)
## S3 method for class 'data.frame'
html(object, file = paste(first.word(deparse(substitute(object))),
                          "html", sep = "."), append = FALSE, link = NULL, linkCol = 1, bgs.col = NULL,
      OutDec=options("OutDec")$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 <code>getwd()</code>
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 <code>factor.names</code> element of <code>design.info(design)</code>
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 <code>OutDec</code> 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 <code>write.csv</code> or <code>write.csv2</code> 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)
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 <code>html</code> , 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
export.design(test, response.names=c("pressure", "temperature"))
```

 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)
```

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. 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. 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	optional seed for the randomization process (integer number)
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. 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 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

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**)

Note

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

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("m","n","o")))
## 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
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)

## 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

Generic and 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 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 from package **conf.design**: [factorize](#)
The function from package **sfsmisc**: [factorize](#)

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))))
```

`fix.design`*Function to preserve class design when editing a design*

Description

This function allows to type in response data and other additional information like date or comments for experimental runs in R, such that all attributes of the design objects are updated accordingly. However, it is recommended to do such editing tasks outside of R, as R is not ideal for these.

Usage

```
### generic function
fix(x, ...)
## Default S3 method:
fix(x, ...)
## S3 method for class 'design'
fix(x, ..., prompt=FALSE)
```

Arguments

<code>x</code>	an object of class <code>design</code>
<code>...</code>	further arguments to function <code>fix</code> from package <code>utils</code>
<code>prompt</code>	logical asking whether user is to be prompted for selecting which new numeric variable is a response; default: all new numeric variables are response variables

Details

Function `fix` is made into an S3 generic. The default method preserves the functionality from package `utils` for all objects that are not of class `design`.

Function `fix` immediately commits all changes, overwriting the original object without further notice. This is also true for the method applicable to objects of class `design`, as long as the changes do not destroy the properties of the design. It is therefore wise to keep a copy of the original, until correctness of the changes is ascertained.

For design objects, changes are only committed, if no forbidden editing has been done: Changes in values of the experimental factors are not allowed and prevent storing edited information (only reported after the fact, user has no choice in the matter).

If the editing would destroy the structure of the design, e.g. because something was typed into an empty row of the data editor which leads to an extra row with missing values, a prompt allows to abort the changes; if changes are committed, the data frame is stored under its original name without any of its attributes (and can thus no longer be treated by any special tools for design objects).

Value

For the default function, refer to `fix` and the links therein.

The function for class `design` adds the numeric version (`as.numeric`) of all newly-entered variables to the `desnum` attribute of `x` and appends the names of all newly-entered response variables to the `response.names` element of the `design.info` attribute.

If numeric variables are to be entered that are no response variables, the function should be called with `prompt=TRUE` so that these are not taken to be response variables. Automatic default analyses (not yet available) would otherwise include analyses for these variables.

Warning

NOTE: Editing the data in R is only helpful for very small examples, especially when using the default editor under Windows.

NEVER do it to the one and only original data file. It does irrevocably replace it in many cases, even if you realize the mistakes you have made before leaving the editor window.

Author(s)

Ulrike Groemping

See Also

See also [fix](#) and the links therein

Examples

```
## Not run:
plan <- oa.design(L18)
fix(plan)
  ## manually add some numeric data for the response
response.names(plan)
fix(plan, prompt=TRUE)
  ## manually add at least two numeric response columns
  ## select from menu which ones are to be responses
response.names(plan)

## End(Not run)
```

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)
```

Arguments

x	an object of class <code>design</code>
...	further arguments to function <code>formula</code>
response	character string giving the name of the response variable (must be among the numeric columns from x) OR integer number giving the position of the response in element <code>response.names</code> of attribute <code>design.info</code>
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 <code>aggregate.design</code> method; this must be an unquoted function name or NULL; 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).

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 be 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 (not implemented yet), the formula will be 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 <code>parseArrayLine</code>
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 <code>n11~f11;n12~f12; ...</code> , where <code>n1</code> stands for the number of levels and <code>f1</code> for their respective frequency; the string may (but need not) contain a trailing semi-colon
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 frequencies 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 frequencies, functions oa.min3, oa.min34, oa.maxGR, oa.maxGR.min34, 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, with.blocks = FALSE, J = FALSE)
contr.XuWu(n, contrasts = TRUE, sparse = FALSE)
oa.min3(ID, nlevels, all, rela = FALSE)
oa.min34(ID, nlevels, min3=NULL, all, rela = FALSE)
oa.max3(ID, nlevels, rela = FALSE)
oa.max4(ID, nlevels, rela = FALSE)
oa.maxGR(ID, nlevels)
oa.maxGR.min34(ID, nlevels, maxGR = NULL)
P3.3(ID, digits = 4, rela=FALSE)
P4.4(ID, digits = 4, rela=FALSE)
GR(ID, digits=2)
nchoosek(n, k)
```

Arguments

design	<p>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 <code>design</code>, which may also include response data.</p> <p>In any case, the design should be a factorial design; the functions are not useful for quantitative designs (like e.g. latin hypercube samples).</p>
with.blocks	<p>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 <code>design</code>.</p> <p>Per default, an existing block factor is ignored.</p> <p>For designs without a block factor, the option does not have an effect.</p> <p>If the design is blocked, and with.blocks is TRUE, the block factor is treated like any other factor in terms of word counting.</p>

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). This is not expected to be useful for practical purposes. 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 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 function contr.XuWu: number of levels of the factor for which to determine contrasts for function nchoosek: number of elements to choose from
contrasts	output a contrast matrix ?
sparse	return a sparse contrast matrix ?
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
min3	the outcome of a call to oa.min3, which is to be used for a call to oa.min34
maxGR	the outcome of a call to oa.maxGR, which is to be used for a call to oa.maxGR.min34
digits	number of decimal points to which to round the result
k	number of elements to be chosen, integer from 0 to n

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). Function `contr.XuWu` provides such 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 sums of all three-factor interaction columns (from a model matrix with all three-factor interactions included), divided by the squared number of runs.

Likewise, the number of generalized words of length 4 is determined by taking the sum of squares of the column sums of all four-factor interaction columns (from a model matrix with all four-factor interactions included), divided by the squared number of runs, 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.

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. The most important ones 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 frequencies 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.

The functions can be used in selecting among different possibilities to accommodate 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, 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`.

Function `GR` calculates the generalized resolution according to Deng and Tang (1999) for 2-level designs or a generalization thereof according to Groemping (2011) for general orthogonal arrays. It returns a value between 3 and 5, where the numeric value 5 stands for “at least 5”.

Functions `oa.min3`, `oa.min34`, `oa.maxGR` and `oa.maxGR.min34` optimize column allocation for a given array for which a certain factor combination must be accommodated: They return designs that allocate columns such that the number of generalized words of length 3 is minimized (`oa.min3`), 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`). Option `rela` allows to switch from the default consideration of absolute numbers of words to relative numbers of words according to Groemping (2011). Function `oa.maxGR` maximizes generalized resolution according to Deng and Tang (1999) as generalized by Groemping (2011), function `oa.maxGR.min34` minimizes relative words of lengths 3 or 4 (as appropriate) among the designs with maximum generalized resolution. 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.)

Value

The functions `length3` and `length4` (currently) per default return the number of 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 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). The function `oa.maxGR` returns a list with elements `GR`, `column.variants`, `RPFTs` and `complete`. Whenever element `GR` is less than 5, element `RPFTs` is a list of the relative projection frequency tables corresponding to the rows of the matrix `column.variants`.

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 `NULL` for `GR=5`.

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

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.

Programming of some of the functions with `rela=TRUE` is not optimized for efficiency; these may take a particularly long time.

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`.

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.beuth-hochschule.de/FB_II/reports/welcome.htm, Department II, Beuth University of Applied Sciences, Berlin.

Xu, H.-Q. and Wu, C.F.J. (2001). Generalized minimum aberration for asymmetrical fractional factorial designs. *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.

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)

## 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.maxGR.min34(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, rela=TRUE)

## 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)

```

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.

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)
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 <code>lm</code> in package stats ; or for the class <code>design</code> method, a data frame of S3 class <code>design</code>
...	further arguments to functions <code>lm</code> , <code>print.lm</code> or <code>print.summary.lm</code>
response	character string giving the name of the response variable (must be among the responses of <code>x</code> ; for wide format repeated measurement or parameter designs, response can also be among the column names of the <code>response</code> element of the <code>design.info</code> attribute) OR integer number giving the position of the response in element <code>response.names</code> of attribute <code>design.info</code>

	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 <code>responsetlist</code> element of the <code>design.info</code> attribute of <code>x</code> .
<code>degree</code>	degree for the formula; if NULL, the default for the formula method is used
<code>FUN</code>	function for the <code>aggregate.design</code> method; this must be an unquoted function name; This option is relevant for repeated measurement designs and parameter designs in long format only
<code>use.center</code>	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 <code>FrF2-package</code> (e.g. function <code>IAPlot</code>)
<code>use.star</code>	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.
<code>projections</code>	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
<code>x</code>	object of class <code>lm</code> or <code>summary.lm</code> , for <code>lm.default</code> like in <code>lm</code>
<code>object</code>	object of class <code>lm.design</code> created by function <code>lm.design</code>
<code>lm.design</code>	a class that is identical in content to class <code>lm</code> ; its purpose is to call a specific print method that provides slightly more detail than the standard printout for linear models
<code>summary.lm.design</code>	a class that is identical in content to class <code>summary.lm</code> ; its purpose is to call a specific print method that provides slightly more detail than the standard summary for linear models
<code>data</code>	like in <code>lm</code>
<code>subset</code>	like in <code>lm</code>
<code>weights</code>	like in <code>lm</code>
<code>na.action</code>	like in <code>lm</code>
<code>method</code>	like in <code>lm</code>
<code>model</code>	like in <code>lm</code>
<code>y</code>	like in <code>lm</code>
<code>qr</code>	like in <code>lm</code>
<code>singular.ok</code>	like in <code>lm</code>
<code>contrasts</code>	like in <code>lm</code>
<code>offset</code>	like in <code>lm</code>

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))

```

Methods for class design objects

Methods for class design objects

Description

Methods for subsetting, printing, summarizing and plotting class design objects

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, ...)
showData(dataframe,colname.bgcolor = "grey50",
          rowname.bgcolor = "grey50",
          body.bgcolor = "white",
          colname.textcolor = "white",
          rowname.textcolor = "white",
          body.textcolor = "black",
          font = "Courier 12",
          maxheight = 30,
          maxwidth = 80,
          title = NULL,
          rowname.bar = "left",
          colname.bar = "top",
          rownumbers = FALSE,
          placement = "-20-40",
          suppress.X11.warnings = TRUE)
## Default S3 method:
showData(dataframe,
          colname.bgcolor = "grey50",

```

```

    rowname.bgcolor = "grey50",
    body.bgcolor = "white",
    colname.textcolor = "white",
    rowname.textcolor = "white",
    body.textcolor = "black",
    font = "Courier 12",
    maxheight = 30,
    maxwidth = 80,
    title = NULL,
    rowname.bar = "left",
    colname.bar = "top",
    rownumbers = FALSE,
    placement = "-20-40",
    suppress.X11.warnings = TRUE)
## S3 method for class 'design'
showData(dataframe,
    colname.bgcolor = "grey50",
    rowname.bgcolor = "grey50",
    body.bgcolor = "white",
    colname.textcolor = "white",
    rowname.textcolor = "white",
    body.textcolor = "black",
    font = "Courier 12",
    maxheight = 30,
    maxwidth = 80,
    title = NULL,
    rowname.bar = "left",
    colname.bar = "top",
    rownumbers = FALSE,
    placement = "-20-40",
    suppress.X11.warnings = TRUE)
## 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)
## S3 method for class 'design'
plot(x, y=NULL, select=NULL, ...)

```

Arguments

x	data frame of S3 class <code>design</code>
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 <code>design</code> , 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 , or in case of plotting, plot , mosaic , or the function plot.design from package graphics
dataframe	data frame; if of S3 class design, the method for design is used, otherwise the default method
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; if NULL, all responses of the design are aggregated; specify names of selected responses (column names of the <code>responselist</code> element of the <code>design.info</code> 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.
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	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;

select has been added in order to obtain manageable plot sizes. For example, mosaic plots are most easily readable 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.

colname.bgcolor	as documented in relimp
rowname.bgcolor	as documented in relimp
body.bgcolor	as documented in relimp
colname.textcolor	as documented in relimp
rowname.textcolor	as documented in relimp
body.textcolor	as documented in relimp
font	as documented in relimp
maxheight	as documented in relimp
maxwidth	as documented in relimp
title	as documented in relimp
rowname.bar	as documented in relimp
colname.bar	as documented in relimp
rownnumbers	as documented in relimp
placement	as documented in relimp
suppress.X11.warnings	as documented in relimp

Details

Items of class [design](#) are data frames with attributes, that have been created for conducting experiments. Apart from the methods documented here, a separate file documents the method [formula.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 default method for showData is the function from package [relimp](#), which is also used in R-Commander for viewing data sets. The method for class design objects is needed for successfully

using the view button of design dataframes in R-commander. They are undesigned before viewing. However, it is preferable to inspect a printout from function `print.design` which provides additional information. This can also be done in the R-Commander using the Design menu in package `RcmdrPlugin.DoE` (to be available soon).

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).

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 `link{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).

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.

Value

extractor The extractor function returns a class design object with modified attributes or a data frame without special attributes, depending on the situation.

If `j` is given, the function always returns a data frame without special attributes, even if `drop.attr=FALSE` or `j=1:ncol(design)`.

If only `i` is given, the default option `drop.attr=TRUE` also returns a data frame without attributes.

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.

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`, `summary` and `plot` 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](#), and function [plot.design](#) from package graphics; a method for function [lm](#) is described in the separate help file [lm.design](#).

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 for plotting designs
  ## 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)
  ## 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 does (columns 1, 14, and 15 of the array) do not occur in
  ## all combinations
  plot(oa.design(L24.2.13.3.1.4.1,nlevels=c(2,2,2,3,4)),select=c("E","D","A"))
  plot(oa.design(L24.2.13.3.1.4.1,nlevels=c(2,2,2,3,4)),select=c("E","D","C"))
```

```

## 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")

#### 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)
origin(ID)
oa
```

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 with option <code>parents.only = TRUE</code> (later it will also be possible to directly access child designs); furthermore, L18, L36 and L54 for the respective Taguchi arrays can be used. Users can also specify names of their own designs here (cf. details). ID must be of class <code>oa</code> . If omitted, ID is automatically determined based on <code>nlevels</code> or <code>factor.names</code> .
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 <code>nlevels</code> is a single number and <code>factor.names</code> is omitted; can otherwise determined from length of <code>factor.names</code> , <code>nlevels</code> or <code>column</code>
nlevels	number(s) of levels, vector with <code>nfactors</code> entries or single number; can be omitted, if obvious from <code>factor.names</code> or if ID and <code>columns</code> are given or if all columns of ID are to be used with default factor names and levels; can be a single number if <code>nfactors</code> is known directly or as length of <code>factor.names</code>
factor.names	a character vector of <code>nfactors</code> factor names or a list with <code>nfactors</code> elements; if the list is named, list names represent factor names, otherwise default factor names are used; the elements of the list are EITHER vectors of appropriate length (corresponding to <code>nlevels</code>) with factor

	<p>levels for the respective factor OR empty strings; Default factor names are the first elements of the character vector <code>Letters</code>, 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.</p>
<code>columns</code>	<p>EITHER a vector of column numbers referring to columns of design ID, assigning a specific column of the array to each factor; this can only be specified, if ID is also given; OR 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”, or “minRelProjAber”. 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 resources, 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.</p>
<code>replications</code>	<p>the number of replications of the array, the setting of <code>repeat.only</code> determines, whether these are real replications or repeated measurements only. Note that replications are not considered for accomodation of <code>min.residual.df</code> residual degrees of freedom, unless a full factorial is used.</p>
<code>repeat.only</code>	<p>default FALSE implies real replications, TRUE implies repeated measurements only</p>
<code>randomize</code>	<p>logical indicating whether the run order is to be randomized ?</p>
<code>seed</code>	<p>integer seed for the random number generator</p>
<code>min.residual.df</code>	<p>minimum number of residual degrees of freedom; Note: function <code>oa.design</code> does not count replications specified with option <code>replications</code> in determining residual degrees of freedom for <code>min.resid.df</code>.</p>

Details

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.)

Currently, package **DoE.base** contains the orthogonal arrays from Warren Kuhfelds collection of “parent” arrays only, plus very few additional designs, and can automatically create the child arrays from Kuhfelds collection, using the replacement instructions provided in the data frame `oacat` through the variable `lineage`. 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.)

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 possible 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 has been implemented. This is not switched on per default because of performance reasons. However, 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. If option `columns="min3"` is chosen, the function tries to allocate the experimental factors to columns of the selected array such that 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). If option `columns="min3.rela"` is chosen, the same approach is taken, but with *relative* number of generalized words according to Groemping (2011). The default `columns="order"` allocates factors from left to right (option `columns="order"`), which is what most software does (but what is not necessarily good). If only very few columns of an array are used, it may be useful to specify the most demanding optimization (which may sometimes take very long) with option `columns="min34"`, which requests that the number of words of generalized length 4 is also minimized (e.g. useful if there are several designs

without any generalized words of length 3). Again, if option `columns="min34.rela"` is chosen, the same approach is taken, but with *relative* number of generalized words according to Groemping (2011). Finally, if option `columns="minRelProjAber"` is chosen, minimum relative projection aberration according to Groemping (2011) is applied ((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 (an example for this will be added shortly)!

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. According to Eric Schoen (personal communication), all the resulting children are isomorphic to each other and are also isomorphic to the Taguchi L18. (This statement holds for qualitative factors only; there are more different possibilities for arrays for quantitative 3 level factors, since arbitrary relabelling of the levels is no longer isomorphic). 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.

Schoen (2009) identified three different isomorphism classes of orthogonal arrays for one 2 level factor and seven 3 level factors. Note that Schoen classifies the isomorphism class of the Taguchi L18 as worst among the three classes, based on the fact that it has more subsets of three factors with bad number of generalized three letter words than the other classes (P3.3 pattern). On the good side, the Taguchi array class also has the most orthogonal subsets of three factors. In most cases, the difference will not be practically relevant.

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:

<code>type</code>	is oa (unless no special orthogonal array is found, in which case a full factorial is created instead, cf. <code>fac.design</code> for its <code>design.info</code> attribute),
<code>nlevels</code>	vector containing the number of levels for each factor
<code>generating.oa</code>	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.

Note

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

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.beuth-hochschule.de/FB_II/reports/welcome.htm, Department II, Beuth University of Applied Sciences, Berlin.

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 <http://support.sas.com/techsup/technote/ts723.html>.

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("mi
## 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 improvement 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[3:8], "comb")
  ## column comb will create the first two columns of the target design
## 6-level design can be created by fac.design or expand.grid
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)

```

```

L18.2.1.3.7.manual <- cbind(nest.des[parent.des[,"comb"],], parent.des)[-9]
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
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)

```

oocat	<i>data frame that lists available orthogonal arrays, mostly from the Kuhfeld collection</i>
-------	--

Description

This data frame holds the list of available orthogonal arrays, except for a few structurally equivalent additional arrays known as Taguchi arrays (L18, L36, L54).

Usage

```
oocat
```

Details

The data frame holds 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 `oocat`. 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 V arrays for which all main effects and 2fis can be orthogonally estimated.

Note that most of the arrays, 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](#)).

Value

The data frame contains the columns `name`, `nruns`, `lineage` and further columns `n2` to `n72`. `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 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 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.

Most of the orthogonal arrays, 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

- Kuhfeld, W. (2009). Orthogonal arrays. Website courtesy of SAS Institute <http://support.sas.com/techsup/technote/ts723.html>.
- Mee, R. (2009). *A Comprehensive Guide to Factorial Two-Level Experimentation*. New York: Springer.

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)
```

param.design	<i>Function to generate Taguchi style parameter designs</i>
--------------	---

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 <code>param.design</code> 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; 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; 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 <code>param.design</code>)
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. <http://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)

```

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 <code>design</code>
quantitative	<p>can be</p> <p>EITHER</p> <p>one of the single entries</p> <p>NA for setting all factors to the default coding for class <code>design</code> (cf. details),</p> <p>“all” for making all factors quantitative (=numeric),</p> <p>“none” for making all factors qualitative (=factor)</p> <p>OR</p> <p>an unnamed vector of length <code>nfactors</code> 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;</p> <p>OR</p> <p>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)</p>
contrasts	<p>only takes effect for factors for which <code>quantitative</code> is FALSE;</p> <p>the default <code>character(0)</code> does not change any contrasts vs.~the previous or default contrasts.</p> <p>For customizing, a</p> <p>character string</p> <p>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</p> <p>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.</p> <p>Possible contrast function names include (at least) <code>contr.FrF2</code> (for number of levels a power of 2 only), <code>contr.helmert</code>, <code>contr.treatment</code>, <code>contr.SAS</code>, <code>contr.sum</code>, <code>contr.poly</code>. CAUTION: Function <code>qua.design</code> checks whether the contrast names actually define a function, but it is not checked whether this function is a valid contrast function.</p>
...	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 qualitative 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 explicitly 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`.
 For function `reptowide`, the design must have repeated measurements (`repeat.only=TRUE` in `design.info` attribute).
 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, specifying optionally specifying selection criteria.

Usage

```
show.oas(name = "all", nruns = "all", nlevels = "all", factors = "all",
         show = 10, parents.only = FALSE)
```

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; 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; nlevels contains the number of levels and number the number of factors for the corresponding number of levels
show	an integer number specifying how many designs are to be listed (upper bound), or the character string "all" for showing all designs, no matter how many. The default is to show 10 designs. show = 0 switches off the display of the result and only returns a value.
parents.only	logical specifying whether to show only parent arrays or child arrays as well; the default is FALSE for inclusion of child arrays

Details

The function shows the arrays that are listed in the data frame `oacat`.

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 design name. The option `parent.only = TRUE` suppresses printing and output of child arrays. The structure of the lineage entry is documented under `oacat`.

Value

A data frame with the three columns `name`, `nruns` and `lineage`, containing the design name, the number of runs and - if applicable - the lineage for generating the design from other designs. The lineage entry is empty for parent designs 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 no design 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 <http://support.sas.com/techsup/technote/ts723.html>.

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

See Also

`oa.design` for using the designs from `oacat` in design creation
`oacat` for the data frame 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)))
## 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 designs by name  
show.oas(name=c("L12.2.11", "L18.2.1.3.7"))
```

SN

*Function for the signal-to-noise ratio $10 * \log_{10}(\text{mean}^2/\text{var})$*

Description

Function for the signal-to-noise ratio $10 * \log_{10}(\text{mean}^2/\text{var})$

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 ($10 * \log_{10}(\text{mean}^2/\text{var})$)

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))
```

Index

*Topic **array**

- add.response, 3
- block.catlg3, 5
- Class design and accessors, 6
- contr.FrF2, 11
- cross.design, 12
- DoE.base-package, 2
- export.design, 14
- fac.design, 17
- factorize, 21
- fix.design, 23
- formula.design, 24
- genChild, 26
- generalized.word.length, 28
- iscube, 34
- lm and aov method for class design objects, 35
- Methods for class design objects, 40
- oa.design, 47
- oocat, 53
- param.design, 55
- qua.design, 57
- Reshape designs with repeated measurements, 60
- show.oas, 62
- SN, 64

*Topic **design**

- add.response, 3
- block.catlg3, 5
- Class design and accessors, 6
- contr.FrF2, 11
- cross.design, 12
- DoE.base-package, 2
- export.design, 14
- fac.design, 17
- factorize, 21
- fix.design, 23
- formula.design, 24

- genChild, 26
- generalized.word.length, 28
- iscube, 34
- lm and aov method for class design objects, 35
- Methods for class design objects, 40
- oa.design, 47
- oocat, 53
- param.design, 55
- qua.design, 57
- Reshape designs with repeated measurements, 60
- show.oas, 62
- SN, 64
- [.design, 10
- [.design(Methods for class design objects), 40

- add.response, 3, 3, 9
- aggregate, 42
- aggregate.data.frame, 44
- aggregate.design, 25, 37, 38, 64
- aggregate.design(Methods for class design objects), 40
- all.equal, 4
- aov, 38
- aov(lm and aov method for class design objects), 35
- aov.design, 38
- apply, 42
- arrays, 53, 55
- as.formula, 25
- as.numeric, 24

- bbd.design, 10
- block.catlg, 19, 20
- block.catlg(block.catlg3), 5
- block.catlg3, 5

- ccd.augment, 10
- ccd.design, 10, 35
- change.contr, 7, 8, 19
- change.contr (qua.design), 57
- Class design and accessors, 6
- class-design-methods (Methods for class design objects), 40
- coef.lm.design (lm and aov method for class design objects), 35
- col.remove (Class design and accessors), 6
- comment, 51
- conf.design, 18
- conf.set, 18, 20
- contr.FrF2, 11, 19, 50, 58
- contr.helmert, 58
- contr.poly, 58
- contr.SAS, 58
- contr.sum, 58
- contr.treatment, 58, 59
- contr.XuWu (generalized.word.length), 28
- contrasts, 12, 30, 42
- cross.design, 3, 10, 12, 56

- DanielPlot, 38
- data frame method, 42
- design, 2–4, 13, 15, 19, 23–25, 28, 29, 36, 38, 39, 41–44, 50, 51, 55, 56, 58, 59
- design (Class design and accessors), 6
- desnum (Class design and accessors), 6
- desnum<- (Class design and accessors), 6
- DoE.base (DoE.base-package), 2
- DoE.base-package, 2
- DoE.wrapper, 3
- DoE.wrapper-package, 16

- export.design, 3–5, 14

- fac.design, 3, 6, 7, 10, 12, 17, 45, 50, 51, 59, 61
- factor.names (Class design and accessors), 6
- factor.names<- (Class design and accessors), 6
- factorize, 18, 21, 22
- fix, 4, 23, 24
- fix (fix.design), 23
- fix.design, 8, 23
- formula, 25, 26
- formula.design, 3, 24, 38, 39, 43
- FrF2, 3, 10, 12, 20, 29, 35, 51, 54, 61
- FrF2-package, 6, 16, 37

- genChild, 26
- generalized.word.length, 2, 3, 28, 54, 55
- getArray (genChild), 26
- GR (generalized.word.length), 28

- html (export.design), 14

- IAPlot, 37, 38
- iscube, 34
- isstar (iscube), 34

- length2 (generalized.word.length), 28
- length3 (generalized.word.length), 28
- length4 (generalized.word.length), 28
- length5 (generalized.word.length), 28
- lengths (generalized.word.length), 28
- Letters, 17, 48
- lhs.design, 10, 59
- lm, 36–38, 45
- lm (lm and aov method for class design objects), 35
- lm and aov method for class design objects, 35
- lm.design, 3, 26, 38, 45

- MEPlot, 38
- Methods for class design objects, 40
- methods for class design objects, 3
- mosaic, 42, 44

- nchoosek (generalized.word.length), 28

- oa, 27
- oa (oa.design), 47
- oa.design, 2, 3, 8, 10, 12, 20, 45, 47, 55, 59, 61, 63
- oa.max3 (generalized.word.length), 28
- oa.max4 (generalized.word.length), 28
- oa.maxGR (generalized.word.length), 28
- oa.min3 (generalized.word.length), 28
- oa.min34 (generalized.word.length), 28
- oa2symb (genChild), 26
- oacat, 27, 29, 49, 53, 63
- ord (Class design and accessors), 6
- origin (oa.design), 47

- P3.3, 50
- P3.3 (generalized.word.length), 28
- P4.4 (generalized.word.length), 28
- param.design, 3, 10, 14, 55
- paramtowide (param.design), 55
- parseArrayLine (genChild), 26
- pb, 10, 12, 20, 35, 51, 61
- pickcube (iscube), 34
- plot, 42
- plot.data.frame, 44
- plot.design, 10, 42, 44, 45
- plot.design (Methods for class design objects), 40
- print, 42
- print.aov.design (lm and aov method for class design objects), 35
- print.design, 10
- print.design (Methods for class design objects), 40
- print.lm, 36
- print.lm.design (lm and aov method for class design objects), 35
- print.summary.aov.design (lm and aov method for class design objects), 35
- print.summary.lm, 36
- print.summary.lm.design (lm and aov method for class design objects), 35

- qua.design, 57

- read.csv, 4
- read.csv2, 4
- redesign (Class design and accessors), 6
- relimp, 43
- reptolong (Reshape designs with repeated measurements), 60
- reptowide (Reshape designs with repeated measurements), 60
- reshape, 9, 61
- Reshape designs with repeated measurements, 60
- response.names (Class design and accessors), 6
- response.names<- (Class design and accessors), 6
- run.order (Class design and accessors), 6
- run.order<- (Class design and accessors), 6
- show.oas, 47, 48, 53, 55, 62
- showData (Methods for class design objects), 40
- SN, 64
- summary, 42
- summary.aov, 38
- summary.aov.design (lm and aov method for class design objects), 35
- summary.design, 10
- summary.design (Methods for class design objects), 40
- summary.lm, 38
- summary.lm.design (lm and aov method for class design objects), 35
- symb2oa (genChild), 26

- undesign (Class design and accessors), 6

- write.csv, 15
- write.csv2, 15

- Yates (block.catlg3), 5
- Yates3 (block.catlg3), 5