Package ‘dae’

December 11, 2014

Version 2.4-0

Date 2014-12-11

Title Functions Useful in the Design and ANOVA of Experiments

Author Chris Brien <Chris.Brien@unisa.edu.au>.

Maintainer Chris Brien <Chris.Brien@unisa.edu.au>

Depends R (>= 2.10.0), ggplot2, methods

Description A number of functions that are useful in manipulating factors,that aid in generating experimental designs and that calculate canonical
efficiency factors for designs. Also there are functions that facilitate
diagnostic checking after an ANOVA, especially when the Error function
has been used in the call to aov.

URL http://chris.brien.name

R topics documented:

`ABC.Interact.dat` *Randomly generated set of values indexed by three factors*

Description

This data set has randomly generated values of the response variable MOE (Measure Of Effectiveness) which is indexed by the two-level factors A, B and C.

Usage

```
data(ABC.Interact.dat)
```

Format

A data.frame containing 8 observations of 4 variables.

Source

Generated by Chris Brien

---

as.numfac *Convert a factor to a numeric vector*

Description

Converts a factor to a numeric vector with approximately the numeric values of its levels. Hence, the levels of the factor must be numeric values, stored as characters. It uses the method described in factor. Use as.numeric to convert the factor to a numeric vector with integers 1, 2, ... corresponding to the positions in the list of levels. You can also use fac.recode to recode the levels to numeric values.

Usage

```
as.numfac(factor)
```

Arguments

factor	The factor to be converted.

Value

A numeric vector. An NA will be stored for any value of the factor whose level is not a number.

Author(s)

Chris Brien

See Also

as.numeric, fac.recode in package **dae**, factor.

Examples

```
## set up a factor and convert it to a numeric vector
a <- factor(rep(1:3, 4))
x <- as.numfac(a)
```

`blockboundary.plot` *This function plots a block boundary on a plot produced by* `design.plot`.

Description

This function plots a block boundary on a plot produced by `design.plot`. It allows control of the starting unit, through rstart and cstart, and the number of rows (nr) and columns (nc) from the starting unit that the blocks to be plotted are to cover.

Usage

```
blockboundary.plot(bdef = NULL, bseq = FALSE, rstart= 0, cstart = 0,
                                nr, nc, bcol = 1, bwd = 2)
```

Arguments

`bdef` A `matrix` of block sizes:

- if there is only one row, then the first element is interpreted as the no. rows in each block and blocks with this number of rows are to be repeated across the rows of the design.
- if there is more than one row, then each row of the matrix specifies a block, with the sequence of rows in the matrix specifying a corresponding sequence of blocks down the rows of the design.

Similarly, a single value for a column specifies a repetition of blocks of that size across the columns of the design, while several column values specifies a sequence of blocks across the columns of the size specified.

`bseq` A `logical` thatt determines whether block numbers are repetitions or sequences of block numbers.

`rstart` A `numeric` speccifying the row after which the plotting of block boundaries is to start.

`cstart` A `numeric` speccifying the column after which the plotting of block boundaries is to start.

`nr` A `numeric` the number of rows (nr), from the starting unit, that the blocks to be plotted are to cover.

`nc` A `numeric` the number of columns (nc), from the starting unit, that the blocks to be plotted are to cover.

`bcol` A `character string` specifying the colour of the block boundary.

See `Colour specification` under the `par` function.

`bwd` A `numeric` giving the width of the block boundary to be plotted.

Value

no values are returned, but modifications are made to the currently active plot.

See Also

`design.plot`, `par`, `DiGGer`

Examples

```
## Not run:
    SPL.Lines.mat <- matrix(as.numfac(Lines), ncol=16, byrow=T)
    colnames(SPL.Lines.mat) <- 1:16
    rownames(SPL.Lines.mat) <- 1:10
    SPL.Lines.mat <- SPL.Lines.mat[10:1, 1:16]
    windows()
    design.plot(SPL.Lines.mat,trts=1:10,new=TRUE,
                rstr="Rows",cstr="Columns", chtdiv=3, rprop = 1,cprop=1,
                plotbndry = TRUE)
    #Plot Mainplot boundaries
    blockboundary.plot(bdef = cbind(4,16), rstart = 1, bwd = 3, bcol = "green",
                       nr = 9, nc = 16)
    blockboundary.plot(bdef = cbind(1,4), bwd = 3, bcol = "green", nr = 1, nc = 16)
    blockboundary.plot(bdef = cbind(1,4), rstart= 9, bwd = 3, bcol = "green",
                       nr = 10, nc = 16)
    #Plot all 4 block boundaries
    blockboundary.plot(bdef = cbind(8,5,5,4), bseq=T, cstart = 1, rstart= 1,
                       bwd = 3,bcol = "blue", nr = 9, nc = 15)
    blockboundary.plot(bdef = cbind(10,16), bwd=3,bcol="blue", nr=10, nc=16)
    #Plot border and internal block boundaries only
    blockboundary.plot(bdef = cbind(8,14), cstart = 1, rstart= 1,
                       bwd = 3, bcol = "blue", nr = 9, nc =  15)
    blockboundary.plot(bdef = cbind(10,16), bwd = 3, bcol = "blue",
                       nr = 10, nc = 16)
## End(Not run)
```

---

correct.degfree — *Check the degrees of freedom in an object of class projector*

Description

Check the degrees of freedom in an object of class "projector".

Usage

```
correct.degfree(object)
```

Arguments

object	An object of class "projector" whose degrees of freedom are to be checked.

Details

The degrees of freedom of the projector are obtained as its number of nonzero eigenvalues. An eigenvalue is regarded as zero if it is less than daeTolerance, which is initially set to 1e-10. The function set.daeTolerance can be used to change daeTolerance.

Value

TRUE or FALSE depending on whether the correct degrees of freedom have been stored in the object of class "projector".

Author(s)

Chris Brien

See Also

`degfree`, `projector` in package **dae**.

`projector` for further information about this class.

Examples

```
## set up a 2 x 2 mean operator that takes the mean of a vector of 2 values
m <- matrix(rep(0.5,4), nrow=2)

## create a projector based on the matrix m
proj.m <- new("projector", data=m)

## add its degrees of freedom
degfree(proj.m) <- 1

## check degrees of freedom are correct
correct.degfree(proj.m)
```

---

decomp.relate — *Examines the relationship between the eigenvectors for two decompositions*

Description

Two decompositions produced by `proj2.decomp` are compared by computing all pairs of crossproduct sums of eigenvectors from the two decompositions. It is most useful when the calls to `proj2.decomp` have the same Q1.

Usage

```
decomp.relate(decomp1, decomp2)
```

Arguments

decomp1	A `list` containing components efficiencies and eigenvectors such as is produced by `proj2.decomp`.
decomp2	Another `list` containing components efficiencies and eigenvectors such as is produced by `proj2.decomp`.

Details

Each element of the r1 x r2 `matrix` is the sum of crossproducts of a pair of eigenvectors, one from each of the two decompositions. A sum is regarded as zero if it is less than `daeTolerance`, which is initially set to 1e-10. The function `set.daeTolerance` can be used to change `daeTolerance`.

Value

A `matrix` that is r1 x r2 where r1 and r2 are the numbers of efficiencies of `decomp1` and `decomp2`, respectively. The `rownames` and `columnnames` of the `matrix` are the values of the efficiency factors from `decomp1` and `decomp2`, respectively.

Author(s)

Chris Brien

See Also

`proj2.decomp`, `proj2.ops` in package **dae**, `eigen`.

Examples

```
## PBIBD(2) from p. 379 of Cochran and Cox (1957) Experimental Designs.
## 2nd edn Wiley, New York
PBIBD2.unit <- list(Blocks = 6, Units = 4)
PBIBD2.nest <- list(Units = "Blocks")
trt <- factor(c(1,4,2,5, 2,5,3,6, 3,6,1,4, 4,1,5,2, 5,2,6,3, 6,3,4,1))
PBIBD2.lay <- fac.layout(unrandomized = PBIBD2.unit,
                         nested.factors=PBIBD2.nest,
                         randomized = trt)

## obtain projectors for units
Q.G <- projector(matrix(1, nrow=24, ncol=24)/24)
Q.B <- projector(fac.meanop(PBIBD2.lay$Blocks) - Q.G)
Q.BP <- projector(diag(1, nrow=24) - Q.B - Q.G)

## obtain projector for trt
Q.T <- projector(fac.meanop(PBIBD2.lay$trt) - Q.G)

## obtain intra- and inter-block decompositions
decomp.inter <- proj2.decomp(Q.B, Q.T)
decomp.intra <- proj2.decomp(Q.BP, Q.T)

## check that intra- and inter-block decompositions are orthogonal
decomp.relate(decomp.intra, decomp.inter)
```

---

degfree *Degrees of freedom extraction and replacement*

Description

Extracts the degrees of freedom from or replaces them in an object of class "`projector`".

Usage

```
degfree(object)

degfree(object) <- value
```

Arguments

`object`	An object of class "`projector`" whose degrees of freedom are to be extracted or replaced.
`value`	An integer to which the degrees of freedom are to be set or an object of class "`projector`" or "matrix" from which the degrees of freedom are to be calulated.

Details

There is no checking of the correctness of the degrees of freedom, either already stored or as a supplied integer value. This can be done using `correct.degfree`.

When the degrees of freedom of the projector are to be calculated, they are obtained as the number of nonzero eigenvalues. An eigenvalue is regarded as zero if it is less than `daeTolerance`, which is initially set to 1e-10. The function `set.daeTolerance` can be used to change `daeTolerance`.

Value

An object of class "`projector`" that consists of a square, summetric, idempotent matrix and degrees of freedom (rank) of the matrix.

Author(s)

Chris Brien

See Also

`correct.degfree`, `projector` in package **dae**.

`projector` for further information about this class.

Examples

```
## set up a 2 x 2 mean operator that takes the mean of a vector of 2 values
m <- matrix(rep(0.5,4), nrow=2)

## coerce to a projector
proj.m <- projector(m)

## extract its degrees of freedom
degfree(proj.m)

## create a projector based on the matrix m
proj.m <- new("projector", data=m)

## add its degrees of freedom and print the projector
degfree(proj.m) <- proj.m
print(proj.m)
```

design.plot	*This function plots treatments within a matrix.*

Description

This function plots treatments within a matrix and may be used to build a graphical representation of a matrix, highlighting the position of certain treatments and the blocking factors used in the design. It is a modified version of the function supplied with DiGGer. It includes more control over the labelling of the rows and columns of the design and allows for more flexible plotting of designs with unequal block size.

Usage

```
design.plot(dsgn, trts = NULL, rprop = 1, cprop = 1, label = TRUE,
                 plotchar = NULL, plotbndry = TRUE, chtdiv = 2,
                 bseq = FALSE, bdef = NULL, bcol = 1, bwd = 2,
                 rotate = FALSE, new = TRUE,
                 cstr = "Range",rstr = "Row", rlab = TRUE, clab = TRUE,
                 font = 1, rdecrease = FALSE, cdecrease = FALSE, ...)
```

Arguments

`dsgn` a `matrix` containing a set of integers or characters representing the treatments.

`trts` A `integer` or `character vector` giving specific treatment labels to be plotted.

`rprop` a value giving the proportion of the row boundary of cell to plot.

`cprop` a value giving the proportion of the column boundary of cell to plot.

`label` a `logical` to indicate whether treatment labels are to be plotted in the cells. If TRUE, print a label for all treatments or specific treatments listed in `trts`. If FALSE, no labels are not printed in the cells.

`plotchar` Either a `character` vector containing labels for the whole set of treatments or a single `integer` specifying a symbol to be used in plotting treatments.

`plotbndry` A `logical` indicting whether a boundary is to plotted around a cell.

`chtdiv` A `numeric` specifying the amount by which plotting text and symbols should be magnified/reduced relative to the default.

`bseq` A `logical` that determines whether block numbers are repetitions or sequences of block numbers.

`bdef` A `matrix` of block sizes:

- if there is only one row, then the first element is interpreted as the no. rows in each block and blocks with this number of rows are to be repeated across the rows of the design.
- if there is more than one row, then each row of the matrix specifies a block, with the sequence of rows in the matrix specifying a corresponding sequence of blocks down the rows of the design.

Similarly, a single value for a column specifies a repetition of blocks of that size across the columns of the design, while several column values specifies a sequence of blocks across the columns of the size specified.

bcol	A character string specifying the colour of the block boundary. See Colour specification under the par function.
bwd	A numeric giving the width of the block boundary to be plotted.
rotate	A logical which, if TRUE, results in the matrix being rotated 90 degrees for plotting.
new	A logical indicating if a new plot is to be produced or the current plot is added to.
cstr	A character string to use as a label for columns of the matrix.
rstr	A character string to use as a label for rows of the matrix.
rlab	A logical indicating each row of the design is labelled. If the rows of the matrix are labelled, these are used; otherwise 1:nrow is used.
clab	A logical indicating each column of the design is labelled. If the columns of the matrix are labelled, these are used; otherwise 1:ncol is used.
font	An integer specifying the font family to be used for row and column labelling.
rdecrease	A logical indicating whether to reverse the row labels.
cdecrease	A logical indicating whether to reverse the column labels.
...	further arguments passed to polygon in plotting the cell.

Value

no values are returned, but a plot is produced.

References

Coombes, N. E. (2009). *DiGGer design search tool in R*. http://www.austatgen.org/files/software/downloads/

See Also

blockboundary.plot, par, polygon, DiGGer

Examples

```
## Not run:
   design.plot(des.mat, trts=1:4, col="lightblue", new=TRUE,
               rstr="Lanes", cstr="Positions", chtdiv=3, rprop = 1,cprop=1,
               plotbndry = TRUE)
   design.plot(des.mat, trts=5:87, label=T, col="grey", chtdiv=3, new=FALSE,
               plotbndry = TRUE)
   design.plot(des.mat, trts=88:434, label=T, col="lightgreen", chtdiv=3,
               new=FALSE, plotbndry = TRUE,
               bseq=TRUE, bdef=cbind(4,10,12), bwd=3, bcol="blue")
## End(Not run)
```

elements — *Extract the elements of an array specified by the subscripts*

Description

Elements of the `array` `x` corresponding to the rows of the two dimensional object `subscripts` are extracted. The number of columns of `subscripts` corresponds to the number of dimensions of `x`. The effect of supplying less columns in `subscripts` than the number of dimensions in `x` is the same as for `"["`.

Usage

```
elements(x, subscripts)
```

Arguments

`x`	An `array` with at least two dimensions whose elements are to be extracted.
`subscripts`	A two dimensional object interpreted as elements by dimensions.

Value

A `vector` containing the extracted elements and whose length equals the number of rows in the `subscripts` object.

See Also

`Extract`

Examples

```
## Form a table of the means for all combinations of Row and Line.
## Then obtain the values corresponding to the combinations in the data frame x,
## excluding Row 3.
x <- fac.gen(list(Row = 2, Line = 4), each =2)
x$y <- rnorm(16)
RowLine.tab <- tapply(x$y, list(x$Row, x$Line), mean)
xs <- elements(RowLine.tab, subscripts=x[x$"Line" != 3, c("Row", "Line")])
```

extab — *Expands the values in table to a vector*

Description

Expands the values in `table` to a `vector` according to the `index.factors` that are considered to index the `table`, either in standard or Yates order. The order of the values in the `vector` is determined by the order of the values of the `index.factors`.

Usage

```
extab(table, index.factors, order="standard")
```

Arguments

table	A numeric vector containing the values to be expanded. Its length must equal the product of the number of used levels for the factors in index.factors and the values in it correspond to all levels combinations of these factors. That is, the values of the index.factors are irrelevant to table.
index.factors	A list of factors that index the table. All the factors must be the same length.
order	The order in which the levels combinations of the index.factors are to be considered as numbered in indexing table; standard numbers them as if they are arranged in standard order, that is with the first factor moving slowest and the last factor moving fastest; yates numbers them as if they are arranged in Yates order, that is with the first factor moving fastest and last factor moving slowest.

Value

A vector of length equal to the factors in index.factor whose values are taken from table.

Author(s)

Chris Brien

Examples

```
## generate a small completely randomized design with the two-level
## factors A and B
n <- 12
CRD.unit <- list(Unit = n)
CRD.treat <- fac.gen(list(A = 2, B = 2), each = 3)
CRD.lay <- fac.layout(unrandomized=CRD.unit, randomized=CRD.treat,
                      seed=956)

## set up a 2 x 2 table of A x B effects
AB.tab <- c(12, -12, -12, 12)

## add a unit-length vector of expanded effects to CRD.lay
attach(CRD.lay)
CRD.lay$AB.effects <- extab(table=AB.tab, index.factors=list(A, B))
```

---

fac.ar1mat *forms the ar1 correlation matrix for a (generalized) factor*

Description

Form the correlation matrix for a (generalized) factor where the correlation between the levels follows an autocorrelation of order 1 (ar1) pattern.

Usage

```
fac.ar1mat(factor, rho)
```

Arguments

factor	The (generalized) factor for which the correlation between its levels displays an ar1 pattern.
rho	The correlation parameter for the ar1 process.

Details

The method is: a) form an n x n matrix of all pairwise differences in the numeric values corresponding to the observed levels of the factor by taking the difference between the following two n x n matrices are equal: 1) each row contains the numeric values corresponding to the observed levels of the factor, and 2) each column contains the numeric values corresponding to the observed levels of the factor, b) replace each element of the pairwise difference matrix with rho raised to the absolute value of the difference.

Value

An n x n matrix, where n is the length of the factor.

Author(s)

Chris Brien

See Also

fac.vcmat, fac.meanop, fac.sumop in package **dae**.

Examples

```
## set up a two-level factor and a three-level factor, both of length 12
A <- factor(rep(1:2, each=6))
B <- factor(rep(1:3, each=2, times=2))

## create a 12 x 12 ar1 matrix corrresponding to B
ar1.B <- fac.ar1mat(B, 0.6)
```

---

fac.combine *Combines several factors into one*

Description

Combines several factors into one whose levels are the combinations of the used levels of the individual factors.

Usage

```
fac.combine(factors, order="standard", combine.levels=FALSE, sep=",", ...)
```

Arguments

factors	A list of factors all of the same length.
order	Either standard or yates. The order in which the levels combinations of the factors are to be considered as numbered when forming the levels of the combined factor; standard numbers them as if they are arranged in standard order, that is with the levels of the first factor moving slowest and those of the last factor moving fastest; yates numbers them as if they are arranged in Yates order, that is with the levels of the first factor moving fastest and those of the last factor moving slowest.
combine.levels	A logical specifying whether the levels labels of the new factor are to be combined from those of the factors being combined. The default is to use the integers from 1 to the product of the numbers of combinations of used levels of the individual factors, numbering the levels according to order.
sep	A character string to separate the levels when combine.levels = TRUE.
...	Further arguments passed to the factor call creating the new factor.

Value

A factor whose levels are formed form the observed combinations of the levels of the individual factors.

Author(s)

Chris Brien

See Also

fac.divide in package **dae**.

Examples

```
## set up two factors
A <- factor(rep(1:2, each=6))
B <- factor(rep(1:3, each=2, times=2))

## obtain six-level factor corresponding to the combinations of A and B
AB <- fac.combine(list(A,B))
```

---

fac.divide *Divides a factor into several individual factors*

Description

Takes a factor and divides it into several individual factors as if the levels in the original factor correspond to the numbering of the levels combinations of the individual factors when these are arranged in standard or Yates order.

Usage

```
fac.divide(combined.factor, factor.names, order="standard")
```

Arguments

combined.factor
: A factor that is to be divided into the individual factors listed in factor.names.

factor.names
: A list of factors to be formed. The names in the list are the names of the factors and the component of a name is either a) a single numeric value that is the number of levels, b) a numeric vector that contains the levels of the factor, or c) a character vector that contains the labels of the levels of the factor.

order
: Either standard or yates. The order in which the levels combinations of the factors in factor.names are to be considered as numbered; standard numbers them as if they are arranged in standard order, that is with the first factor moving slowest and the last factor moving fastest; yates numbers them as if they are arranged in Yates order, that is with the first factor moving fastest and last factor moving slowest.

Value

A data.frame whose columns consist of the factors listed in factor.names and whose values have been computed from the combined factor. All the factors will be of the same length.

Note

A single factor name may be supplied in the list in which case a data.frame is produced that contains the single factor computed from the numeric vector. This may be useful when calling this function from others.

Author(s)

Chris Brien

See Also

fac.combine in package **dae**.

Examples

```
## generate a small completely randomized design for 6 treatments
n <- 12
CRD.unit <- list(Unit = n)
treat <- factor(rep(1:4, each = 3))
CRD.lay <- fac.layout(unrandomized=CRD.unit, randomized=treat, seed=956)

## divide the treatments into two two-level factor A nd B
CRD.facs <- fac.divide(CRD.lay$treat, factor.names = list(A = 2, B = 2))
```

fac.gen *Generate all combinations of several factors*

Description

Generate all combinations of several factors.

Usage

```
fac.gen(generate, each=1, times=1, order="standard")
```

Arguments

`generate`	A `list` of named objects and numbers that specify the `factors` whose `levels` are to be generated and the pattern in these `levels`.If a component of the `list` is named, then the component should be either a) a single `numeric` value that is the number of `levels`, b) a numeric `vector` that contains the `levels` of the `factor`, or c) a character `vector` that contains the `labels` of the `levels` of the `factor`.
`each`	The number of times to replicate consecutively the elements of the `levels` generated according to pattern specified by the `generate` argument.
`times`	The number of times to repeat the whole generated pattern of `levels` generated according to pattern specified by the `generate` argument.
`order`	Either `standard` or `yates`. The order in which the speed of cycling through the `levels` is to move; combinations of the `factors` are to be considered as numbered; `standard` cycles through the levels of the first factor slowest and the last factor moving fastest; `yates` cycles through the levels of the first factor fastest and last factor moving slowest.

Details

The `levels` of each `factor` are generated in a hierarchical pattern where the `levels` of one `factor` are held constant while those of the adjacent `factor` are cycled through the complete set once. If a number is supplied instead of a name, the pattern is generated as if a `factor` with that number of `levels` had been supplied in the same position as the number. However, no `levels` are stored for this unamed `factor`.

Value

A `data.frame` of generated `levels` with columns corresponding to the codefactors in the `generate` list.

Warning

Avoid using factor names F and T as these might be confused with FALSE and TRUE.

Author(s)

Chris Brien

See Also

fac.combine in package **dae**

Examples

```
## generate a 2^3 factorial experiment with levels - and +, and
## in Yates order
mp <- c("-", "+")
fnames <- list(Catal = mp, Temp = mp, Press = mp, Conc = mp)
Fac4Proc.Treats <- fac.gen(generate = fnames, order="yates")

## Generate the factors A, B and D. The basic pattern has 4 repetitions
## of the levels of D for each A and B combination and 3 repetitions of
## the pattern of the B and D combinations for each level of A. This basic
## pattern has each combination repeated twice, and the whole of this
## is repeated twice. It generates 864 A, B and D combinations.
gen <- list(A = 3, 3, B = c(0,100,200), 4, D = c("0","1"))
fac.gen(gen, times=2, each=2)
```

---

fac.layout *Generate a randomized layout for an experiment*

Description

Generate a layout for an experiment consisting of randomized factors that are randomized to the unrandomized factors, taking into account the nesting, for the design, between the unrandomized factors.

Usage

```
fac.layout(unrandomized, nested.factors=NULL, randomized, seed=NULL)
```

Arguments

unrandomized
: A data.frame or a list of factors, along with their levels. If a list, the name of each component of the list is a factor name and the component is either a single numeric value that is the number of levels, a numeric vector that contains the levels of the factor or a character codevector that contains the labels of the levels of the factor.

nested.factors
: A list of the unrandomized factors that are nested in other factors in unrandomized. The name of each component is the name of a factor that is nested and the component is a character vector containing the factors within which it is nested. It is emphasized that the nesting is a property of the design that is being employed (it is only partly based on the intrinsic nesting.

randomized
: A factor or a data.frame containing the values of the factor(s) to be randomized.

seed
: A single value, interpreted as an integer, that specifies the starting value of the random number generator.

Details

This functon uses the method of randomization described by Bailey (1981). That is, a permutation of the units that respects the nesting for the design is obtained. This permutation is applied jointly to the unrandomized and randomized factors to produce the randomized layout. The Units and Permutation vectors enable one to swap between this permutation and the randomized layout.

Value

A data.frame consisting of the values for Units and Permutation vectors along with the values for the unrandomized and randomized factors that specify the randomized layout for the experiment.

Author(s)

Chris Brien

References

Bailey, R.A. (1981) A unified approach to design of experiments. *Journal of the Royal Statistical Society, Series A*, **144**, 214–223.

See Also

fac.gen in package **dae**.

Examples

```
## generate a randomized layout for a 4 x 4 Latin square
## (the nested.factors agument is not needed here as none of the
## factors are nested)
LS.unit <- data.frame(row = ordered(rep(c("I","II","III","IV"), times=4)),
                      col = factor(rep(c(0,2,4,6), each=4)))
LS.ran <- data.frame(treat = factor(c(1:4, 2,3,4,1, 3,4,1,2, 4,1,2,3)))
data.frame(LS.unit, LS.ran)
LS.lay <- fac.layout(unrandomized=LS.unit, randomized=LS.ran, seed=7197132)
LS.lay[LS.lay$Permutation,]

## generate a randomized layout for a replicated randomized complete
## block design, with the block factors arranged in standard order for
## rep then plot and then block
RCBD.unit <- list(rep = 2, plot=1:3, block = c("I","II"))
## specify that plot is nested in block and rep and that block is nested
## in rep
RCBD.nest <- list(plot = c("block","rep"), block="rep")
## generate treatment factor in systematic order so that they correspond
## to plot
tr <- factor(rep(1:3, each=2, times=2))
## obtain randomized layout
RCBD.lay <- fac.layout(unrandomized=RCBD.unit,
                       nested.factors=RCBD.nest,
                       randomized=tr, seed=9719532)
#sort into the original standard order
RCBD.perm <- RCBD.lay[RCBD.lay$Permutation,]
#resort into randomized order
RCBD.lay <- RCBD.perm[order(RCBD.perm$Units),]
```

```
## generate a layout for a split-unit experiment in which:
## - the main-unit factor is A with 4 levels arranged in
##   a randomized complete block design with 2 blocks;
## - the split-unit factor is B with 3 levels.
SPL.unit <- list(block = 2, main.unit = 4, split.unit = 3)
SPL.nest <- list(main.unit = "block", split.unit = c("block", "main.unit"))
SPL.trt <- fac.gen(list(A = 4, B = 3), times = 2)
SPL.lay <- fac.layout(unrandomized=SPL.unit,
                      nested.factors=SPL.nest,
                      randomized=SPL.trt, seed=155251978)
```

`fac.match` *Match, for each combination of a set of columns in* `x`*, the row that has the same combination in* `table`

Description

Match, for each combination of a set of columns in `x`, the row that has the same combination in `table`. That is, there must be only one row in `table` for each combination of the specified set of columns in `x`. It can be viewed as a generalization of the match function from a single vector to multiple vectors.

Usage

```
fac.match(x, table, col.names)
```

Arguments

`x`	an R object, normally a `data.frame`, possibly a `matrix`.
`table`	an R object, normally a `data.frame`, possibly a `matrix`.
`col.names`	A `character vector` giving the columns in `x` and `table` that are to be matched.

Value

A `vector` of length equal to `x` that gives the rows in `table` that match the combinations of `col.names` in `x`.

Author(s)

Chris Brien

See Also

`match`

Examples

```
## Not run:
index <- fac.match(adj.water.dat, longi.dat,  c("Cart","Days"))
longi.dat[index, c("Weight.Before", "Weight.After")] <-
                    adj.water.dat[,c("Weight.Before", "Weight.After")]

## End(Not run)
```

`fac.meanop` *computes the projection matrix that produces means*

Description

Computes the symmetric projection matrix that produces the means corresponding to a (generalized) `factor`.

Usage

```
fac.meanop(factor)
```

Arguments

`factor`	The (generalized) `factor` whose means the projection matrix computes from an observation-length vector.

Details

The design matrix **X** for a (generalized) `factor` is formed with a column for each `level` of the (generalized) `factor`, this column being its indicator variable. The projection matrix is formed as `X %*% (1/diag(r) %*% t(X)`, where `r` is the `vector` of `levels` replications.

A generalized `factor` is a `factor` formed from the combinations of the `levels` of several original `factors`. Generalized `factors` can be formed using `fac.combine`.

Value

A `projector` containing the symmetric, projection matrix and its degrees of freedom.

Author(s)

Chris Brien

See Also

`fac.combine`, `projector`, `degfree`, `correct.degfree`, `fac.sumop` in package **dae**.

`projector` for further information about this class.

Examples

```
## set up a two-level factor and a three-level factor, both of length 12
A <- factor(rep(1:2, each=6))
B <- factor(rep(1:3, each=2, times=2))

## create a generalized factor whose levels are the combinations of A and B
AB <- fac.combine(list(A,B))

## obtain the operator that computes the AB means from a vector of length 12
M.AB <- fac.meanop(AB)
```

`fac.nested` *creates a factor whose values are generated within those of the factor nesting.fac*

Description

Creates a `factor` whose `levels` are generated within those of the factor `nesting.fac`. All elements of `nesting.fac` having the same level are numbered from 1 to the number of different elements having that level.

Usage

```
fac.nested(nesting.fac, levels=NA, labels=NA, ...)
```

Arguments

`nesting.fac`	The `factor` within each of whose `levels` the created `factor` is to be generated.
`levels`	Optional `vector` of levels for the `factor`. Any data value that does not match a value in `levels` will be `NA` in the `factor`. The default value of `levels` is the the list of numbers from 1 to the maximum replication of the `levels` of `nesting.fac`, represented as characters.
`labels`	Optional `vector` of values to use as `labels` for the `levels` of the `factor`. The default is `as.character(levels)`.
`...`	Further arguments passed to the `factor` call dreating the new `factor`.

Value

A `factor` that is a character `vector` with class attribute "`factor`" and a `levels` attribute which determines what character strings may be included in the `vector`.

Note

The levels of `nesting.fac` do not have to be equally replicated.

Author(s)

Chris Brien

See Also

`fac.gen` in package **dae**, `factor`.

Examples

```
## set up factor A
A <- factor(c(1, 1, 1, 2, 2))

## create nested factor
B <- fac.nested(A)
```

`fac.recode` *Recodes the `levels` and values of a factor using the value in position i of the `newlevels` vector to replace the ith `level` of the `factor`.*

Description

Recodes factor `levels` using values in a vector. The new levels do not have to be unique.

Usage

```
fac.recode(factor, newlevels, ...)
```

Arguments

`factor`	The `factor` to be recoded.
`newlevels`	A `vector` of length `levels(factor)` containing values to use in the recoding.
`...`	Further arguments passed to the `factor` call dreating the new `factor`.

Value

A `factor`.

Author(s)

Chris Brien

See Also

`as.numfac` and `mpone` in package **dae**, `factor`, `relevel`.

Examples

```
## set up a factor with labels
a <- factor(rep(1:4, 4), labels=c("A","B","C","D"))

## recode "A" and "D" to 1 and "B" and "C" to 2
b <- fac.recode(a, c(1,2,2,1))
```

`fac.sumop` *computes the summation matrix that produces sums corresponding to a factor*

Description

Computes the matrix that produces the sums corresponding to a (generalized) `factor`.

Usage

```
fac.sumop(factor)
```

Arguments

factor — The (generalized) factor whose sums the summation matrix computes from an observation-length vector.

Details

The design matrix **X** for a (generalized) factor is formed with a column for each level of the (generalized) factor, this column being its indicator variable. The summation matrix is formed as X %*% t(X).

A generalized factor is a factor formed from the combinations of the levels of several original factors. Generalized factors can be formed using fac.combine.

Value

A symmetric matrix.

Author(s)

Chris Brien

See Also

fac.combine, fac.meanop in package **dae**.

Examples

```
## set up a two-level factoir and a three-level factor, both of length 12
A <- factor(rep(1:2, each=6))
B <- factor(rep(1:3, each=2, times=2))

## create a generlaized factor whose levels are the combinations of A and B
AB <- fac.combine(list(A,B))

## obtain the operator that computes the AB means from a vector of length 12
S.AB <- fac.sumop(AB)
```

---

fac.vcmat — *forms the variance matrix for the variance component of a (generalized) factor*

Description

Form the variance matrix for a (generalized) factor whose effects for its different levels are independently and identically distributed, with their variance given by the variance component; elements of the matrix will equal either zero or sigma2 and displays compound symmetry.

Usage

```
fac.vcmat(factor, sigma2)
```

Arguments

factor	The (generalized) factor for which the varaince matrix is required.
sigma2	The variance component, being the of the random effects for the factor.

Details

The method is: a) form the n x n summation or relationship matrix whose elements are equal to zero except for those elements whose corresponding elements in the following two n x n matrices are equal: 1) each row contains the numeric values corresponding to the observed levels of the factor, and 2) each column contains the numeric values corresponding to the observed levels of the factor, b) multiply the summation matrix by sigma2.

Value

An n x n matrix, where n is the length of the factor.

Author(s)

Chris Brien

See Also

fac.ar1mat, fac.meanop, fac.sumop in package **dae**.

Examples

```
## set up a two-level factor and a three-level factor, both of length 12
A <- factor(rep(1:2, each=6))
B <- factor(rep(1:3, each=2, times=2))

## create a 12 x 12 ar1 matrix corrresponding to B
vc.B <- fac.vcmat(B, 2)
```

Fac4Proc.dat *Data for a 2^4 factorial experiment*

Description

The data set come from an unreplicated 2^4 factorial experiment to investigate a chemical process. The response variable is the Conversion percentage (Conv) and this is indexed by the 4 two-level factors Catal, Temp, Press and Conc, with levels “-” and “+”. The data is aranged in Yates order. Also included is the 16-level factor Runs which gives the order in which the combinations of the two-level factors were run.

Usage

```
data(Fac4Proc.dat)
```

Format

A data.frame containing 16 observations of 6 variables.

Source

Table 10.6 of Box, Hunter and Hunter (1978) *Statistics for Experimenters*. New York, Wiley.

`fitted.aovlist` *Extract the fitted values for a fitted model from an aovlist object*

Description

Extracts the fitted values as the sum of the effects for all the fitted terms in the model, stopping at `error.term` if this is specified. It is a method for the generic function `fitted`.

Usage

```
## S3 method for class aovlist
fitted(object, error.term=NULL, ...)
```

Arguments

`object`	An `aovlist` object created from a call to `aov`.
`error.term`	The term from the `Error` function down to which effects are extracted for adding to the fitted values. The order of terms is as given in the ANOVA table. If `error.term` is `NULL` effects are extracted from all `Error` terms.
`...`	Further arguments passed to or from other methods.

Value

A numeric `vector` of fitted values.

Note

Fitted values will be the sum of effects for terms from the model, but only for terms external to any `Error` function. If you want effects for terms in the `Error` function to be included, put them both inside and outside the `Error` function so they are occur twice.

Author(s)

Chris Brien

See Also

`fitted.errors`, `resid.errors`, `tukey.1df` in package **dae**.

Examples

```
## set up data frame for randomized complete block design in Table 4.4 from
## Box, Hunter and Hunter (2005) Statistics for Experimenters. 2nd edn
## New York, Wiley.
RCBDPen.dat <- fac.gen(list(Blend=5, Flask=4))
RCBDPen.dat$Treat <- factor(rep(c("A","B","C","D"), times=5))
RCBDPen.dat$Yield <- c(89,88,97,94,84,77,92,79,81,87,87,
                       85,87,92,89,84,79,81,80,88)
```

```
## perform the analysis of varaince
RCBDPen.aov <- aov(Yield ~ Blend + Treat + Error(Blend/Flask), RCBDPen.dat)
summary(RCBDPen.aov)

## two equivalent ways of extracting the fitted values
fit  <- fitted.aovlist(RCBDPen.aov)
fit <- fitted(RCBDPen.aov, error.term = "Blend:Flask")
```

`fitted.errors` *Extract the fitted values for a fitted model*

Description

An alias for the generic function `fitted`. When it is available, the method `fitted.aovlist` extracts the fitted values, which is provided in the **dae** package to cover `aovlist` objects.

Usage

```
## S3 method for class errors
fitted(object, ...)
```

Arguments

`object`	An `object` for which the extraction of model fitted values is meaningful.
`...`	Further arguments passed to or from other methods.

Value

A numeric vector of fitted values.

Warning

See `fitted.aovlist` for specific information about fitted values when an `Error` function is used in the call to the `aov` function.

Author(s)

Chris Brien

See Also

`fitted.aovlist`, `resid.errors`, `tukey.1df` in package **dae**.

Examples

```
## set up data frame for randomized complete block design in Table 4.4 from
## Box, Hunter and Hunter (2005) Statistics for Experimenters. 2nd edn
## New York, Wiley.
RCBDPen.dat <- fac.gen(list(Blend=5, Flask=4))
RCBDPen.dat$Treat <- factor(rep(c("A","B","C","D"), times=5))
RCBDPen.dat$Yield <- c(89,88,97,94,84,77,92,79,81,87,87,
                       85,87,92,89,84,79,81,80,88)
```

```
## perform the analysis of varaince
RCBDPen.aov <- aov(Yield ~ Blend + Treat + Error(Blend/Flask), RCBDPen.dat)
summary(RCBDPen.aov)

## three equivalent ways of extracting the fitted values
fit  <- fitted.aovlist(RCBDPen.aov)
fit <- fitted(RCBDPen.aov, error.term = "Blend:Flask")
fit <- fitted.errors(RCBDPen.aov, error.term = "Blend:Flask")
```

`get.daeTolerance` *Gets the value of daeTolerance for the package dae*

Description

A function that gets the value such that, in **dae** functions, values less than it are considered to be zero.

Usage

```
get.daeTolerance()
```

Value

The value of daeTolerance.

Author(s)

Chris Brien

See Also

`set.daeTolerance`.

Examples

```
## get daeTolerance.
get.daeTolerance()
```

`interaction.ABC.plot` *Plots an interaction plot for three factors*

Description

Plots a `function` (the mean by default) of the `response` for the combinations of the three `factors` specified as the `x.factor` (plotted on the x axis of each plot), the `groups.factor` (plotted as separate lines in each plot) and the `trace.factor` (its `levels` are plotted in different plots). Interaction plots for more than three `factors` can be produced by using `fac.combine` to combine all but two of them into a single `factor` that is specified as the `trace.factor`.

Usage

```
interaction.ABC.plot(response, x.factor, groups.factor,
       trace.factor,data, fun="mean", title="A:B:C Interaction Plot",
       xlab, ylab, key.title, lwd=4, columns=2, ...)
```

Arguments

`response`	A numeric `vector` containing the response variable from which a function (the mean by default) is computed for plotting on the y-axis.
`x.factor`	The `factor` to be plotted on the x-axis of each plot.
`groups.factor`	The `factor` plotted as separate lines in each plot.
`trace.factor`	The `factor` for whose `levels` there are separate plots.
`data`	A `data.frame` containing the three `factors` and the `response`.
`fun`	The `function` to be computed from the `response` for each combination of the three factors `x.factor`, `groups.factor` and `trace.factor`. By default, the mean is computed for each combination.
`title`	Title for plot window. By default it is "A:B:C Interaction Plot".
`xlab`	`Label` for the x-axis. By default it is the name of the `x.factor`.
`ylab`	`Label` for the y-axis. By default it is the name of the `response`.
`key.title`	`Label` for the xkey to the lines in each plot. By default it is the name of the `groups.factor`.
`lwd`	The width of the `lines`. By default it is 4.
`columns`	The number of columns for arranging the several plots for the levels of the `groups.factor`. By default it is 2.
`...`	Other arguments that are passed down to the function `xyplot`.

Value

An object of class "`trellis`", which is automatically plotted by `print.trellis`.

Note

A `data.frame` called `data.means` is created, attached and detached during execution of this function.

Author(s)

Chris Brien

See Also

`fac.combine` in package **dae**, `interaction.plot`.

Examples

```
## plot for generated data
## use ?ABC.Interact.dat for data set details
data(ABC.Interact.dat)
interaction.ABC.plot(MOE, A, B, C, data=ABC.Interact.dat)

## plot for Example 14.1 from Mead, R. (1990). The Design of Experiments:
## Statistical Principles for Practical Application. Cambridge,
## Cambridge University Press.
## use ?SPLGrass.dat for details
data(SPLGrass.dat)
interaction.ABC.plot(Main.Grass, x.factor=Period,
                     groups.factor=Spring, trace.factor=Summer,
                     data=SPLGrass.dat,
                     title="Effect of Period, Spring and Summer on Main Grass")
```

---

is.allzero — *Tests whether all elements are approximately zero*

Description

A single-line `function` that tests whether all elements are zero (approximately).

Usage

```
is.allzero(x)
```

Arguments

x	An `object` whose elements are to be tested.

Details

All the elements of `x` are tested as being less than `daeTolerance`, which is initially set to 1e-10. The function `set.daeTolerance` can be used to change `daeTolerance`.

Value

A `logical`.

Author(s)

Chris Brien

Examples

```
## create a vector of 9 zeroes and a one
y <- c(rep(0,9), 1)

## check that vector is only zeroes is FALSE
is.allzero(y)
```

`is.projector` *Tests whether an object is a valid object of class projector*

Description

Tests whether an `object` is a valid object of class "`projector`".

Usage

```
is.projector(object)
```

Arguments

`object`	The `matrix` to be made into a projector.

Details

The function `is.projector` tests whether the object consists of a `matrix` that is square, symmetric and idempotent. In checking symmetry and idempotency, the equality of the matrix with either its transpose or square is tested. In this, a difference in elements is considered to be zero if it is less than `daeTolerance`, which is initially set to 1e-10. The function `set.daeTolerance` can be used to change `daeTolerance`.

Value

`TRUE` or `FALSE` depending on whether the object is a valid object of class "`projector`".

Warning

The degrees of freedom are not checked. `correct.degfree` can be used to check them.

Author(s)

Chris Brien

See Also

`projector`, `correct.degfree` in package **dae**.

`projector` for further information about this class.

Examples

```
## set up a 2 x 2 mean operator that takes the mean of a vector of 2 values
m <- matrix(rep(0.5,4), nrow=2)

## create an object of class projector
proj.m <- projector(m)

## check that it is a valid projector
is.projector(proj.m)
```

`mat.ar1` *Forms an ar1 correlation matrix*

Description

Form the correlation `matrix` of order `order` whose correlations follow the ar1 pattern. The `matrix` has diagonal elements equal to one and the off-diagonal element in the ith row and jth column equal to ρ^k where $k = |i - j|$.

Usage

```
mat.ar1(order, rho)
```

Arguments

`order`	The order of the `matrix` to be formed.
`rho`	The correlation on the first off-diagonal.

Value

A correlation `matrix` whose elements follow an ar1 pattern.

See Also

`mat.I`, `mat.J`

Examples

```
corr <- mat.ar1(order=4, rho=0.4)
```

`mat.dirprod` *Forms the direct product of two matrices*

Description

Form the direct product of the $m \times n$ `matrix` **A** and the $p \times q$ `matrix` **B**. It is also called the Kroneker product and the right direct product. It is defined to be the result of replacing each element of **A**, a_{ij}, with $a_{ij}\boldsymbol{B}$. The result `matrix` is $mp \times nq$.

The method employed uses the `rep` function to form two $mp \times nq$ matrices: (i) the direct product of **A** and **J**, and (ii) the direct product of **J** and **B**, where each **J** is a matrix of ones whose dimensions are those required to produce an $mp \times nq$ matrix. Then the elementwise product of these two matrices is taken to yield the result.

Usage

```
mat.dirprod(A, B)
```

Arguments

A	The left-hand matrix in the product.
B	The right-hand matrix in the product.

Value

An $mp \times nq$ matrix.

See Also

matmult

Examples

```
col.I <- mat.I(order=4)
row.I <- mat.I(order=28)
V <- mat.dirprod(col.I, row.I)
```

---

mat.I *Forms a unit matrix*

Description

Form the unit or identity matrix of order order.

Usage

```
mat.I(order)
```

Arguments

order	The order of the matrix to be formed.

Value

A square matrix whose diagonal elements are one and its off-diagonal are zero.

See Also

mat.J, mat.ar1

Examples

```
col.I <- mat.I(order=4)
```

mat.J — *Forms a square matrix of ones*

Description

Form the square matrix of ones of order order.

Usage

```
mat.J(order)
```

Arguments

order	The order of the matrix to be formed.

Value

A square matrix all of whose elements are one.

See Also

mat.I, mat.ar1

Examples

```
col.J <- mat.J(order=4)
```

meanop — *computes the projection matrix that produces means*

Description

Replaced by fac.meanop.

mpone — *Converts the first two levels of a factor into the numeric values -1 and +1*

Description

Converts the first two levels of a factor into the numeric values -1 and +1.

Usage

```
mpone(factor)
```

Arguments

factor	The factor to be converted.

Value

A numeric vector.

Warning

If the factor has more than two levels they will be coerced to numeric values.

Author(s)

Chris Brien

See Also

mpone in package **dae**, factor, relevel.

Examples

```
## generate all combinations of two two-level factors
mp <- c("-", "+")
Frf3.trt <- fac.gen(list(A = mp, B = mp))

## add factor C, whose levels are the products of the levles of A and B
Frf3.trt$C <- factor(mpone(Frf3.trt$A)*mpone(Frf3.trt$B), labels = mp)
```

---

no.reps *Computes the number of replicates for an experiment*

Description

Computes the number of pure replicates required in an experiment to achieve a specified power.

Usage

```
no.reps(multiple=1., df.num=1.,
        df.denom=expression((df.num + 1.) * (r - 1.)), delta=1.,
        sigma=1., alpha=0.05, power=0.8, tol=0.025, print=FALSE)
```

Arguments

multiple	The multiplier, m, which when multiplied by the number of pure replicates of a treatment, r, gives the number of observations rm used in computing means for some, not necessarily proper, subset of the treatment factors; m is the replication arising from other treatment factors. However, for single treatment factor experiments the subset can only be the treatment factor and m = 1.
df.num	The degrees of freedom of the numerator of the F for testing the term involving the treatment factor subset.
df.denom	The degrees of freedom of the denominator of the F for testing the term involving the treatment factor subset.
delta	The true difference between a pair of means for some, not necessarily proper, subset of the treatment factors.
sigma	The population standard deviation.

alpha	The significance level to be used.
power	The minimum power to be achieved.
tol	The maximum difference tolerated between the power required and the power computed in determining the number of replicates.
print	TRUE or FALSE to have or not have a table of power calculation details printed out.

Value

A single numeric value containing the computed number of pure replicates.

Author(s)

Chris Brien

See Also

power.exp in package **dae**.

Examples

```
## Compute the number of replicates (blocks) required for a randomized
## complete block design with four treatments.
no.reps(multiple = 1, df.num = 3,
         df.denom = expression(df.num * (r - 1)), delta = 5,
       sigma = sqrt(20), print = TRUE)
```

---

power.exp *Computes the power for an experiment*

Description

Computes the power for an experiment.

Usage

```
power.exp(rm=5., df.num=1., df.denom=10., delta=1., sigma=1.,
                 alpha=0.05, print=FALSE)
```

Arguments

rm	The number of observations used in computing a mean.
df.num	The degrees of freedom of the numerator of the F for testing the term involving the means.
df.denom	The degrees of freedom of the denominator of the F for testing the term involving the means.
delta	The true difference between a pair of means.
sigma	The population standard deviation.
alpha	The significance level to be used.
print	TRUE or FALSE to have or not have a table of power calculation details printed out.

Value

A single numeric value containing the computed power.

Author(s)

Chris Brien

See Also

no.reps in package **dae**.

Examples

```
## Compute power for a randomized complete block design with four treatments
## and five blocks.
rm <- 5
power.exp(rm = rm, df.num = 3, df.denom = 3 * (rm - 1), delta = 5,
          sigma = sqrt(20),print = TRUE)
```

---

print.projector *Print projectors*

Description

Print an object of class "projector", displaying the matrix and its degrees of freedom (rank).

Usage

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

Arguments

x	The object of class "projector" to be printed.
...	Further arguments passed to or from other methods.

Author(s)

Chris Brien

See Also

print, print.default, show.

projector for further information about this class.

Examples

```
## set up a 2 x 2 mean operator that takes the mean of a vector of 2 values
m <- matrix(rep(0.5,4), nrow=2)

## create an object of class projector
proj.m <- projector(m)

## print the object either using the Method function, the generic function or show
print.projector(proj.m)
print(proj.m)
proj.m
```

---

proj2.decomp *Canonical efficiency factors and eigenvectors in joint decomposition of two projectors*

Description

Computes the canonical efficiency factors for the joint decomposition of two projection matrices and the eigenvectors corresponding to the first projector (James and Wilkinson, 1971).

Usage

```
proj2.decomp(Q1, Q2)
```

Arguments

Q1	An object of class "projector".
Q2	An object of class "projector".

Details

The component efficiencies is a `vector` containing the nonzero canonical efficiency factors for the joint decomposition of the two projectors. The nonzero canonical efficiency factors are the nonzero eigenvalues of Q1 %*% Q2 %*% Q1 (James and Wilkinson, 1971). An eigenvalue is regarded as zero if it is less than `daeTolerance`, which is initially set to 1e-10. The function `set.daeTolerance` can be used to change `daeTolerance`.

The component eigenvectors is an n x r `matrix`, where n is the order of the projectors and r is the number of nonzero canonical efficiency factors; it contains the eigenvectors of Q1 corresponding to the nonzero canonical efficiency factors. The eigenvectors for Q2 can be obtained by premultiplying those for Q1 by Q2.

Value

A `list` with components efficiencies and eigenvectors.

Author(s)

Chris Brien

References

James, A. T. and Wilkinson, G. N. (1971) Factorization of the residual operator and canonical decomposition of nonorthogonal factors in the analysis of variance. *Biometrika*, **58**, 279-294.

See Also

`proj2.efficiency`, `proj2.ops` in package **dae**, `eigen`.

`projector` for further information about this class.

Examples

```
## PBIBD(2) from p. 379 of Cochran and Cox (1957) Experimental Designs.
## 2nd edn Wiley, New York
PBIBD2.unit <- list(Blocks = 6, Units = 4)
PBIBD2.nest <- list(Units = "Blocks")
trt <- factor(c(1,4,2,5, 2,5,3,6, 3,6,1,4, 4,1,5,2, 5,2,6,3, 6,3,4,1))
PBIBD2.lay <- fac.layout(unrandomized = PBIBD2.unit,
                         nested.factors=PBIBD2.nest,
                         randomized = trt)

## obtain projectors for units
Q.G <- projector(matrix(1, nrow=24, ncol=24)/24)
Q.B <- projector(fac.meanop(PBIBD2.lay$Blocks) - Q.G)
Q.BP <- projector(diag(1, nrow=24) - Q.B - Q.G)

## obtain projector for trt
Q.T <- projector(fac.meanop(PBIBD2.lay$trt) - Q.G)

## obtain intra- and inter-block decompositions
decomp.inter <- proj2.decomp(Q.B, Q.T)
decomp.intra <- proj2.decomp(Q.BP, Q.T)

#extract intrablock efficiencies
decomp.intra$efficiencies
```

`proj2.efficiency` *Computes the canonical efficiency factors for the joint decomposition of two projection matrices*

Description

Computes the canonical efficiency factors for the joint decomposition of two projection matrices (James and Wilkinson, 1971).

Usage

```
proj2.efficiency(Q1, Q2)
```

Arguments

Q1	An object of class "`projector`".
Q2	An object of class "`projector`".

Details

The nonzero canonical efficiency factors are the nonzero eigenvalues of Q1 %*% Q2 %*% Q1 (James and Wilkinson, 1971). An eigenvalue is regarded as zero if it is less than `daeTolerance`, which is initially set to 1e-10. The function `set.daeTolerance` can be used to change `daeTolerance`.

Value

A `vector` containing the nonzero canonical efficiency factors.

Author(s)

Chris Brien

References

James, A. T. and Wilkinson, G. N. (1971) Factorization of the residual operator and canonical decomposition of nonorthogonal factors in the analysis of variance. *Biometrika*, **58**, 279-294.

See Also

`proj2.decomp`, `proj2.ops` in package **dae**, `eigen`.

`projector` for further information about this class.

Examples

```
## PBIBD(2) from p. 379 of Cochran and Cox (1957) Experimental Designs.
## 2nd edn Wiley, New York
PBIBD2.unit <- list(Blocks = 6, Units = 4)
PBIBD2.nest <- list(Units = "Blocks")
trt <- factor(c(1,4,2,5, 2,5,3,6, 3,6,1,4, 4,1,5,2, 5,2,6,3, 6,3,4,1))
PBIBD2.lay <- fac.layout(unrandomized = PBIBD2.unit,
                         nested.factors=PBIBD2.nest,
                         randomized = trt)

## obtain projectors for units
Q.G <- projector(matrix(1, nrow=24, ncol=24)/24)
Q.B <- projector(fac.meanop(PBIBD2.lay$Blocks) - Q.G)
Q.BP <- projector(diag(1, nrow=24) - Q.B - Q.G)

## obtain projector for trt
Q.T <- projector(fac.meanop(PBIBD2.lay$trt) - Q.G)

## produce interblock efficiencies
proj2.efficiency(Q.B, Q.T)

## save intrablock efficiencies
eff.intra <- proj2.efficiency(Q.BP, Q.T)
```

proj2.ops *Compute the projection and Residual operators for two, possibly nonorthogonal, projection matrices*

Description

A procedure that computes the projection operators that decompose the range of Q1 into a part that pertains to Q2 and a part that is orthogonal to Q2. It also produces the nonzero canonical efficiency factors for the joint decomposition of Q1 and Q and the corresponding eigenvectors of Q1 (James and Wilkinson, 1971). Q1 and Q2 may be nonorthogonal.

Usage

```
proj2.ops(Q1, Q2)
```

Arguments

Q1	A symmetric `projector` whose range is to be decomposed.
Q2	A symmetric `projector` whose range in Q1 is required.

Details

The nonzero canonical efficiency factors are the nonzero eigenvalues of Q1 %*% Q2 %*% Q1 (James and Wilkinson, 1971). An eigenvalue is regarded as zero if it is less than `daeTolerance`, which is initially set to 1e-10. The function `set.daeTolerance` can be used to change `daeTolerance`.

The eigenvectors are the eigenvectors of Q1 corresponding to the nonzero canonical efficiency factors. The eigenvectors for Q2 can be obtained by premultiplying those for Q1 by Q2.

Qres is computed using equation 4.10 from James and Wilkinson (1971) and Qconf is obtained by subtracting Qres from Q1.

Value

A `list` with the following components:

1. **efficiencies:** a `vector` containing the nonzero canonical efficiency factors;
2. **eigenvectors:** an n x r `matrix`, where n is the order of the projectors and r is the number of nonzero canonical efficiency factors; it contains the eigenvectors of Q1 corresponding to the nonzero canonical efficiency factors.
3. **Qconf:** a `projector` onto the part of the range of Q1 with which Q2 is confounded;
4. **Qres:** a `projector` onto the part of the range of Q1 that is orthogonal to the range of Q2.

Author(s)

Chris Brien

References

James, A. T. and Wilkinson, G. N. (1971) Factorization of the residual operator and canonical decomposition of nonorthogonal factors in the analysis of variance. *Biometrika*, **58**, 279-294.

See Also

proj2.decomp, proj2.efficiency, decomp.relate in package **dae**.

projector for further information about this class.

Examples

```
## PBIBD(2) from p. 379 of Cochran and Cox (1957) Experimental Designs.
## 2nd edn Wiley, New York
PBIBD2.unit <- list(Blocks = 6, Units = 4)
PBIBD2.nest <- list(Units = "Blocks")
trt <- factor(c(1,4,2,5, 2,5,3,6, 3,6,1,4, 4,1,5,2, 5,2,6,3, 6,3,4,1))
PBIBD2.lay <- fac.layout(unrandomized = PBIBD2.unit,
                         nested.factors=PBIBD2.nest,
                         randomized = trt)

## obtain projectors for units
Q.G <- projector(matrix(1, nrow=24, ncol=24)/24)
Q.B <- projector(fac.meanop(PBIBD2.lay$Blocks) - Q.G)
Q.BP <- projector(diag(1, nrow=24) - Q.B - Q.G)

## obtain projector for trt
Q.T <- projector(fac.meanop(PBIBD2.lay$trt) - Q.G)

## obtain the projection operators for the interblock analysis
PBIBD2.Bops <- proj2.ops(Q.B, Q.T)
Q.B.T <- PBIBD2.Bops$Qconf
Q.B.res <- PBIBD2.Bops$Qres

## demonstrate their orthogonality
is.allzero(Q.B.T %*% Q.B.res)
```

---

projector *Create projectors*

Description

The class "projector" is the subclass of the class "matrix" in which matrices are square, symmetric and idempotent.

The function projector tests whether a matrix satisfies these criteria and if it does creates a "projector" object, computing the projector's degrees of freedom and adding them to the object.

Usage

```
projector(Q)
```

Arguments

Q	The matrix to be made into a projector.

Details

In checking that the matrix is square, symmetric and idempotent, the equality of the matrix with either its transpose or square is tested. In this, a difference in elements is considered to be zero if it is less than daeTolerance, which is initially set to 1e-10. The function set.daeTolerance can be used to change daeTolerance.

Value

An object of Class "projector" that consists of a square, summetric, idempotent matrix and degrees of freedom (rank) of the matrix.

Author(s)

Chris Brien

See Also

degfree, correct.degfree in package **dae**.

projector for further information about this class.

Examples

```
## set up a 2 x 2 mean operator that takes the mean of a vector of 2 values
m <- matrix(rep(0.5,4), nrow=2)

## create an object of class projector
proj.m <- projector(m)

## check that it is a valid projector
is.projector(proj.m)
```

---

projector-class *Class projector*

Description

The class "projector" is the subclass of matrices that are square, symmetric and idempotent.

is.projector is the membership function for this class.

degfree is the extractor function for the degrees of freedom and degfree<- is the replacement function.

correct.degfree checks whether the stored degrees of freedom are correct.

Objects from the Class

An object of class "projector" consists of a square, symmetric, idempotent matrix along with its degrees of freedom (rank).

Objects can be created by calls of the form `new("projector", data, nrow, ncol, byrow, dimnames, ...)`. However, this does not add the degrees of freedom to the object. These can be added using the replacement function degfree<-. Alternatively, the function projector creates the new object from a matrix, adding its degrees of freedom at the same time.

Slots

`.Data`: Object of class "matrix"

`degfree`: Object of class "integer"

Extends

Class "`matrix`", from data part. Class "`array`", by class "matrix", distance 2. Class "`structure`", by class "matrix", distance 3. Class "`vector`", by class "matrix", distance 4, with explicit coerce.

Methods

coerce `signature(from = "projector", to = "matrix")`

print `signature(x = "projector")`

show `signature(object = "projector")`

Author(s)

Chris Brien

See Also

`projector`, `degfree`, `correct.degfree` in package **dae**.

Examples

```
showClass("projector")

## set up a 2 x 2 mean operator that takes the mean of a vector of 2 values
m <- matrix(rep(0.5,4), nrow=2)

## create an object of class projector
proj.m <- projector(m)

## check that it is a valid projector
is.projector(proj.m)

## create a projector based on the matrix m
proj.m <- new("projector", data=m)

## add its degrees of freedom and print the projector
degfree(proj.m) <- proj.m
```

---

qqyeffects *Half or full normal plot of Yates effects*

Description

Produces a half or full normal plot of the Yates effects from a 2^k factorial experiment.

Usage

```
qqyeffects(aov.obj, error.term="Within", data=NULL, pch=16,
           full=FALSE, ...)
```

Arguments

aov.obj	An aov object or aovlistobject created from a call to aov.
error.term	The term from the Error function from which the Yates effects are estimated. Only required when Error used in call to aov.
data	A data.frame in which the variables specified in the aov.obj will be found. If missing, the variables are searched for in the standard way.
pch	The number of a plotting symbol to be drawn when plotting points (use help(points) for details).
full	whether a full or half normal plot is to be produced. The default is for a half-normal plot; full=TRUE produces a full normal plot.
...	Further graphical parameters may be specified (use help(par) for possibilities.

Details

A half or full normal plot of the Yates effects is produced. You will be able to interactively select effects to be labelled (click reasonably close to the point and on the side where you want the label placed). **Right click on the graph and select Stop when you have finished labelling effects.** A regression line fitted to the unselected effects and constrained to go through the origin is plotted. Also, a list of the labelled effects, if any, are printed to standard ouptut.

Value

Returns, invisibly, a list with components x and y, giving coordinates of the plotted points.

Author(s)

Chris Brien

See Also

yates.effects in package **dae**, qqnorm.

Examples

```
## analysis of 2^4 factorial experiment from Table 10.6 of Box, Hunter and
## Hunter (1978) Statistics for Experimenters. New York, Wiley.
## use ?Fac4Proc.dat for data set details
data(Fac4Proc.dat)
Fac4Proc.aov <- aov(Conv ~ Catal * Temp * Press * Conc + Error(Runs),
                                                         Fac4Proc.dat)
qqyeffects(Fac4Proc.aov, error.term="Runs", data=Fac4Proc.dat)
```

---

resid.errors *Extract the residuals for a fitted model*

Description

An alias for the generic function residuals. When it is available, the method residuals.aovlist extracts residuals, which is provided in the package **dae** to cover aovlist objects.

Usage

```
resid.errors(object, ...)
```

Arguments

object	An object for which the extraction of residuals is meaningful.
...	Further arguments passed to or from other methods.

Value

A numeric vector containing the residuals.

Note

See residuals.aovlist for specific information about the residuals when an Error function is used in the call to the aov function.

Author(s)

Chris Brien

See Also

fitted.errors, residuals.aovlist, tukey.1df in package **dae**.

Examples

```
## set up data frame for randomized complete block design in Table 4.4 from
## Box, Hunter and Hunter (2005) Statistics for Experimenters. 2nd edn
## New York, Wiley.
RCBDPen.dat <- fac.gen(list(Blend=5, Flask=4))
RCBDPen.dat$Treat <- factor(rep(c("A","B","C","D"), times=5))
RCBDPen.dat$Yield <- c(89,88,97,94,84,77,92,79,81,87,87,
                       85,87,92,89,84,79,81,80,88)

## perform the analysis of varaince
RCBDPen.aov <- aov(Yield ~ Blend + Treat + Error(Blend/Flask), RCBDPen.dat)
summary(RCBDPen.aov)

## two equivalent ways of extracting the residuals
res  <- residuals.aovlist(RCBDPen.aov)
res <- residuals(RCBDPen.aov, error.term = "Blend:Flask")
res <- resid.errors(RCBDPen.aov)
```

---

residuals.aovlist *Extract the residuals from an aovlist object*

Description

Extracts the residuals from error.term or, if error.term is not specified, the last error.term in the analysis. It is a method for the generic function residuals.

Usage

```
## S3 method for class aovlist
residuals(object, error.term=NULL, ...)
```

Arguments

`object`	An `aovlist` object created from a call to `aov`.
`error.term`	The term from the `Error` function for which the residuals are to be extracted. If `error.term` is `NULL` the residuals are extracted from the last `Error` term.
`...`	Further arguments passed to or from other methods.

Value

A numeric `vector` containing the residuals.

Author(s)

Chris Brien

See Also

`fitted.errors`, `resid.errors`, `tukey.1df` in package **dae**.

Examples

```
## set up data frame for randomized complete block design in Table 4.4 from
## Box, Hunter and Hunter (2005) Statistics for Experimenters. 2nd edn
## New York, Wiley.
RCBDPen.dat <- fac.gen(list(Blend=5, Flask=4))
RCBDPen.dat$Treat <- factor(rep(c("A","B","C","D"), times=5))
RCBDPen.dat$Yield <- c(89,88,97,94,84,77,92,79,81,87,87,
                       85,87,92,89,84,79,81,80,88)

## perform the analysis of varaince
RCBDPen.aov <- aov(Yield ~ Blend + Treat + Error(Blend/Flask), RCBDPen.dat)
summary(RCBDPen.aov)

## two equivalent ways of extracting the residuals
res  <- residuals.aovlist(RCBDPen.aov)
res <- residuals(RCBDPen.aov, error.term = "Blend:Flask")
```

`rmvnorm` *generates a vector of random values from a multivariate normal distribution*

Description

Generates a vector of random values from an n-dimensional multivariate normal distribution whose mean is given by the n-vector `mean` and variance by the n x n symmetric matrix `V`. It uses the method described by Ripley (1987, p.98)

Usage

```
rmvnorm(mean, V, method = eigenanalysis)
```

Arguments

`mean`	The mean vector of the multivariate normal distribution from which the random values are to be generated.
`V`	The variance matrix of the multivariate normal distribution from which the random values are to be generated.
`method`	The method used to decompose the variance matrix in producing a a matrix to transform the iid standard normal values. The two methods available are `eigenanalysis` and `choleski`, where only the first letter of each option is obligatory. The default `method` is eigenanalysis, which is slower but is likely to be more stable than Choleski decomposition.

Details

The method is: a) uses either the eigenvalue or Choleski decomposition of the variance matrix, `V`, to form the matrix that transforms an iid vector of values to a vector with variance `V`; b) generate a vector of length equal to `mean` of standard normal values; c) premultiply the vector of standard normal values by the transpose of the upper triangular factor and, to the result, add `mean`.

Value

A `vector` of length n, equal to the length of `mean`.

Author(s)

Chris Brien

References

Ripley, B. D. (1987) *Stochastic simulation*. Wiley, New York.

See Also

`fac.ar1mat`, `fac.vcmat`, in package **dae**, `rnorm`, and `chol`.

Examples

```
## set up a two-level factor and a three-level factor, both of length 12
A <- factor(rep(1:2, each=6))
B <- factor(rep(1:3, each=2, times=2))

## generate random values from a multivariate normal for which
#the mean is 20 for all variables and
#the variance matrix has random effects for factor A, ar1 pattern for B and
#residual random variation
mean <- rep(20, 12)
V <- fac.vcmat(A, 5) + fac.ar1mat(B, 0.6) + 2*mat.I(12)
y <- rmvnorm(mean, V)
```

set.daeTolerance *Sets the value of daeTolerance for the package dae*

Description

A function that sets the value such that, in **dae** functions, values less than it are considered to be zero. Initially, daeTolerance is set to 1e-10.

Usage

```
set.daeTolerance(tolerance)
```

Arguments

tolerance	The value to which daeTolerance is to be set.

Value

The value of daeTolerance is returned invisibly.

Author(s)

Chris Brien

See Also

get.daeTolerance.

Examples

```
## set daeTolerance.
set.daeTolerance(.Machine$double.eps ^ 0.5)
```

show-methods *Methods for Function show in Package dae*

Description

Methods for function show in Package **dae**

Methods

signature(object = "projector") Prints the matrix and its degrees of freedom.

See Also

projector for further information about this class.

`SPLGrass.dat` *Data for an experiment to investigate the effects of grazing patterns on pasture composition*

Description

The response variable is the percentage area covered by the principal grass (Main.Grass). The design for the experiment is a split-unit design. The main units are arranged in 3 Rows x 3 Columns. Each main unit is split into 2 SubRows x 2 SubColumns.

The factor Period, with levels 3, 9 and 18 days, is assigned to the main units using a 3 x 3 Latin square. The two-level factors Spring and Summer are assigned to split-units using a criss-cross design within each main unit. The levels of each of Spring and Summer are two different grazing patterns in its season.

Usage

```
data(SPLGrass.dat)
```

Format

A data.frame containing 36 observations of 8 variables.

Source

Example 14.1 from Mead, R. (1990). *The Design of Experiments: Statistical Principles for Practical Application.* Cambridge, Cambridge University Press.

`strength` *Generate paper strength values*

Description

Generates paper strength values for an experiment with different temperatures.

Usage

```
strength(nodays, noruns, temperature, ident)
```

Arguments

`nodays`	The number of days over which the experiment is to be run.
`noruns`	The number of runs to be performed on each day of the experiment.
`temperature`	A `factor` that encapsulates the layout by giving the temperature to be investigated for each run on each day. These must be ordered so that the temperatures for the first day are given in the order in which they are to be investigated on that day. These must be followed by the noruns temperatures for the second day and so on. Consequently, the factor temperature will have nodays*noruns values.
`ident`	The digits of your student identity number. That is, leave out any letters.

Value

A `data.frame` object containing the `factors` day, run and temperature and a `vector` of the generated strengths.

Author(s)

Chris Brien

Examples

```
## Here temperature is a factor with 4*3 = 12 values whose
## first 3 values specify the temperatures to be applied in
## the 3 runs on the first day, values 4 to 6 specify the
## temperatures for the 3 runs on day 2, and so on.
temperature <- factor(rep(c(80,85,90), 4))
exp.strength <- strength(nodays = 4, noruns = 3,
                         temperature = temperature, ident = 0123456)

## In this second example, a completely randomized design is generated
## for the same 3 temperatures replicated 4 times. The layout is stored
## in the data.frame called Design.
Design <- fac.layout(unrandomized=list(runs = 12),
                     randomized = temperature,
                     seed = 5847123)
## eradicate the unrandomized version of temperature
remove("temperature")

## The 12 temperatures in Design are to be regarded as being assigned to
## days and runs in the same manner as for the first example.
exp.strength <- strength(nodays = 4, noruns = 3,
                         temperature = Design$temperature, ident = 0123456)
```

`tukey.1df` *Performs Tukey's one-degree-of-freedom-test-for-nonadditivity*

Description

Performs Tukey's one-degree-of-freedom-test-for-nonadditivity on a set of residuals from an analysis of variance.

Usage

```
tukey.1df(aov.obj, data, error.term="Within")
```

Arguments

`aov.obj`	An aov object or `aovlist` object created from a call to `aov`.
`error.term`	The term from the `Error` function whose residuals are to be tested for nonadditivity. Only required when the `Error` function used in call to aov, so that an `aovlist` object is created.
`data`	A `data.frame` containing the original response variable and factors used in the call to `aov`.

Value

A `list` containing Tukey.SS, Tukey.F, Tukey.p, Devn.SSq being the SSq for the 1df test, F value for test and the p-value for the test.

Note

In computing the test quantities fitted values must be obtained. If `error.term` is specified, fitted values will be the sum of effects extracted from terms from the `Error` function, but only down to that specified by `error.term`.The order of terms is as given in the ANOVA table. If `error.term` is unspecified, all effects for terms external to any `Error` terms are extracted and summed.

Extracted effects will only be for terms external to any `Error` function. If you want effects for terms in the `Error` function to be included, put them both inside and outside the `Error` function so they are occur twice.

Author(s)

Chris Brien

See Also

`fitted.errors`, `resid.errors` in package **dae**.

Examples

```
## set up data frame for randomized complete block design in Table 4.4 from
## Box, Hunter and Hunter (2005) Statistics for Experimenters. 2nd edn
## New York, Wiley.
RCBDPen.dat <- fac.gen(list(Blend=5, Flask=4))
RCBDPen.dat$Treat <- factor(rep(c("A","B","C","D"), times=5))
RCBDPen.dat$Yield <- c(89,88,97,94,84,77,92,79,81,87,87,
                       85,87,92,89,84,79,81,80,88)

## perform the analysis of varaince
RCBDPen.aov <- aov(Yield ~ Blend + Treat + Error(Blend/Flask), RCBDPen.dat)
summary(RCBDPen.aov)

## Obtain the quantities for Tukeys test
tukey.1df(RCBDPen.aov, RCBDPen.dat, error.term = "Blend:Flask")
```

`yates.effects` *Extract Yates effects*

Description

Extracts Yates effects from an aov object or `aovlist` object.

Usage

```
yates.effects(aov.obj, error.term="Within", data=NULL)
```

Arguments

- `aov.obj` — An `aov` object or `aovlist` object created from a call to `aov`.
- `error.term` — The term from the `Error` function from which the Yates effects are estimated. Only required when `Error` used in call to `aov`.
- `data` — A `data.frame` in which the variables specified in the `aov.obj` will be found. If missing, the variables are searched for in the standard way.

Details

Yates effects are specific to 2^k experiments, where Yates effects are conventionally defined as the difference between the upper and lower levels of a factor. We follow the convention used in Box, Hunter and Hunter (1978) for scaling of higher order interactions: all the Yates effects are on the same scale, and represent the average difference due to the interaction between two different levels. Effects are estimated only from the error term supplied to the `error.term` argument.

Value

A `vector` of the Yates effects.

Author(s)

Chris Brien

See Also

`qqyeffects` in package **dae**, `aov`.

Examples

```
## analysis of 2^4 factorial experiment from Table 10.6 of Box, Hunter and
## Hunter (1978) Statistics for Experimenters. New York, Wiley.
## use ?Fac4Proc.dat for data set details
data(Fac4Proc.dat)
Fac4Proc.aov <- aov(Conv ~ Catal * Temp * Press * Conc + Error(Runs),
                                                    Fac4Proc.dat)
round(yates.effects(Fac4Proc.aov, error.term="Runs", data=Fac4Proc.dat), 2)
```

Index