Package ‘dae’

January 12, 2020

Version 3.1-21
Date 2020-01-12
Title Functions Useful in the Design and ANOVA of Experiments
Depends R (>= 3.5.0), ggplot2
Imports graphics, methods, plyr, stats
Suggests testthat, R.rsp
VignetteBuilder R.rsp

Description The content falls into the following groupings: (i) Data, (ii) Factor manipulation functions, (iii) Design functions, (iv) ANOVA functions, (v) Matrix functions, (vi) Projector and canonical efficiency functions, and (vii) Miscellaneous functions. There is a vignette describing how to use the design functions for randomizing and assessing designs available as a vignette called 'DesignNotes'. The ANOVA functions facilitate the extraction of information when the 'Error' function has been used in the call to 'aov'. The package 'dae' can also be installed from <http://chris.brien.name/rpackages/>.

License GPL (>= 2)
URL http://chris.brien.name
RoxygenNote 5.0.1
NeedsCompilation no
Author Chris Brien [aut, cre] (<https://orcid.org/0000-0003-0581-1817>)
Maintainer Chris Brien <chris.brien@unisa.edu.au>
Repository CRAN
Date/Publication 2020-01-12 10:40:02 UTC

R topics documented:
daef-package ................................................................. 4
ABC.Interact.dat ............................................................. 8
as.data.frame.pstructure .................................................. 9
R topics documented:

as.numfac ................................................................. 10
BIBD.Wheat.dat ......................................................... 11
blockboundaryPlot .................................................... 11
Cabinet1.des ............................................................ 13
Casuarina.dat .......................................................... 14
correct.degfree ........................................................ 14
daedeprecated .......................................................... 15
daetips ................................................................. 16
decomp.relate ........................................................... 16
degfree ................................................................. 18
designAmeasures ......................................................... 19
designAnatomy ........................................................... 21
designBlocksGGPlot .................................................... 24
designGGPlot ............................................................ 27
designLatinSqrSys ....................................................... 30
designPlot ............................................................... 30
designPlotlabels .......................................................... 33
designRandomize ......................................................... 35
designTwoPhaseAnatomies ............................................. 38
detect.diff ............................................................. 40
efficiencies .............................................................. 41
efficiency.criteria ....................................................... 43
elements ................................................................. 44
Exp249.munit.des ....................................................... 45
extab ................................................................. 46
fac.ar1mat ............................................................... 47
fac.combine ............................................................ 48
fac.divide .............................................................. 49
fac.gen ................................................................. 50
fac.match ............................................................... 52
fac.meanop ............................................................. 53
fac.multinested .......................................................... 54
fac.nested .............................................................. 56
fac.recode ............................................................. 57
fac.sumop .............................................................. 58
fac.vcmat .............................................................. 59
Fac4Proc.dat ............................................................ 60
fitted.aovlist ........................................................... 60
fitted.errors ............................................................. 62
get.daeTolerance ......................................................... 63
harmonic.mean ........................................................... 64
interaction.ABC.plot .................................................. 64
is.allzero ............................................................... 66
is.projector ............................................................. 67
LatticeSquare_t49.des ................................................... 68
marginality ............................................................... 68
mat.ar1 ................................................................. 70
mat.ar2 ................................................................. 70
### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>mat.ar3</td>
<td>71</td>
</tr>
<tr>
<td>mat.arma</td>
<td>73</td>
</tr>
<tr>
<td>mat.banded</td>
<td>74</td>
</tr>
<tr>
<td>mat.dirprod</td>
<td>75</td>
</tr>
<tr>
<td>mat.dirsum</td>
<td>76</td>
</tr>
<tr>
<td>mat.exp</td>
<td>77</td>
</tr>
<tr>
<td>mat.gau</td>
<td>77</td>
</tr>
<tr>
<td>mat.I</td>
<td>78</td>
</tr>
<tr>
<td>mat.J</td>
<td>79</td>
</tr>
<tr>
<td>mat.ma1</td>
<td>80</td>
</tr>
<tr>
<td>mat.ma2</td>
<td>80</td>
</tr>
<tr>
<td>mat.ncssvar</td>
<td>81</td>
</tr>
<tr>
<td>mat.sar</td>
<td>82</td>
</tr>
<tr>
<td>mat.sar2</td>
<td>83</td>
</tr>
<tr>
<td>mat.Vpred</td>
<td>84</td>
</tr>
<tr>
<td>mat.Vpredicts</td>
<td>86</td>
</tr>
<tr>
<td>McIntyreTMV.dat</td>
<td>89</td>
</tr>
<tr>
<td>meanop</td>
<td>90</td>
</tr>
<tr>
<td>mpone</td>
<td>90</td>
</tr>
<tr>
<td>no.reps</td>
<td>91</td>
</tr>
<tr>
<td>Oats.dat</td>
<td>92</td>
</tr>
<tr>
<td>p2canon.object</td>
<td>93</td>
</tr>
<tr>
<td>pcanon.object</td>
<td>94</td>
</tr>
<tr>
<td>power.exp</td>
<td>95</td>
</tr>
<tr>
<td>print.aliasing</td>
<td>96</td>
</tr>
<tr>
<td>print.projector</td>
<td>97</td>
</tr>
<tr>
<td>print.pstructure</td>
<td>98</td>
</tr>
<tr>
<td>print.summary.p2canon</td>
<td>99</td>
</tr>
<tr>
<td>print.summary.pcanon</td>
<td>100</td>
</tr>
<tr>
<td>proj2.combine</td>
<td>101</td>
</tr>
<tr>
<td>proj2.efficiency</td>
<td>102</td>
</tr>
<tr>
<td>proj2.eigen</td>
<td>104</td>
</tr>
<tr>
<td>projector</td>
<td>105</td>
</tr>
<tr>
<td>projector-class</td>
<td>106</td>
</tr>
<tr>
<td>projs.2canon</td>
<td>108</td>
</tr>
<tr>
<td>projs.combine.p2canon</td>
<td>109</td>
</tr>
<tr>
<td>pstructure.formula</td>
<td>110</td>
</tr>
<tr>
<td>pstructure.object</td>
<td>114</td>
</tr>
<tr>
<td>qqyeffects</td>
<td>115</td>
</tr>
<tr>
<td>resid.errors</td>
<td>116</td>
</tr>
<tr>
<td>residuals.aovlist</td>
<td>117</td>
</tr>
<tr>
<td>rmvnorm</td>
<td>118</td>
</tr>
<tr>
<td>Sensory3Phase.dat</td>
<td>119</td>
</tr>
<tr>
<td>set.daeTolerance</td>
<td>120</td>
</tr>
<tr>
<td>show-methods</td>
<td>121</td>
</tr>
<tr>
<td>SPLGrass.dat</td>
<td>122</td>
</tr>
<tr>
<td>strength</td>
<td>122</td>
</tr>
<tr>
<td>summary.p2canon</td>
<td>124</td>
</tr>
</tbody>
</table>
dae-package

Functions Useful in the Design and ANOVA of Experiments

Description

The content falls into the following groupings: (i) Data, (ii) Factor manipulation functions, (iii) Design functions, (iv) ANOVA functions, (v) Matrix functions, (vi) Projector and canonical efficiency functions, and (vii) Miscellaneous functions. There is a vignette describing how to use the design functions for randomizing and assessing designs available as a vignette called 'DesignNotes'. The ANOVA functions facilitate the extraction of information when the 'Error' function has been used in the call to 'aov'. The package 'dae' can also be installed from <http://chris.brien.name/rpackages/>.

Version: 3.1-21
Date: 2020-01-12

Index

(i) Data

<table>
<thead>
<tr>
<th>Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC.Interact.dat</td>
<td>Randomly generated set of values indexed by three factors</td>
</tr>
<tr>
<td>BIBD.Wheat.dat</td>
<td>Data for a balanced incomplete block experiment</td>
</tr>
<tr>
<td>Casuarina.dat</td>
<td>Data for an experiment with rows and columns from Williams (2002)</td>
</tr>
<tr>
<td>Exp249.munit.des</td>
<td>Systematic, main-plot design for an experiment to be run in a greenhouse</td>
</tr>
<tr>
<td>Fac4Proc.dat</td>
<td>Data for a 2^4 factorial experiment</td>
</tr>
<tr>
<td>LatticeSquare_t49.des</td>
<td>A Lattice square design for 49 treatments</td>
</tr>
<tr>
<td>McIntyreTMV.dat</td>
<td>The design and data from McIntyre (1955) two-phase experiment</td>
</tr>
<tr>
<td>Oats.dat</td>
<td>Data for an experiment to investigate nitrogen response of 3 oats varieties</td>
</tr>
<tr>
<td>Sensory3PhaseShort.dat</td>
<td>Data for the three-phase sensory evaluation experiment in Brien, C.J. and Payne, R.W. (1999), but with short factor names</td>
</tr>
<tr>
<td>SPLGrass.dat</td>
<td>Data for an experiment to investigate the effects of grazing patterns on pasture composition</td>
</tr>
</tbody>
</table>
(ii) Factor manipulation functions

-as.numfac  Convert a factor to a numeric vector
-fac.combine  Combines several factors into one
-fac.divide  Divides a factor into several individual factors
-fac.gen  Generate all combinations of several factors and, optionally, replicate them
-fac.match  Match, for each combination of a set of columns in 'x', the row that has the same combination in 'table'
-fac.multinested  Creates several factors, one for each level of a nesting.fac and each of whose values are either generated within those of the level of nesting.fac or using the values of a nested.fac
-fac.nested  Creates a factor, the nested factor, whose values are generated within those of a nesting factor
-fac.recode  Recodes factor 'levels' using possibly nonunique values in a vector.
-mpone  Converts the first two levels of a factor into the numeric values -1 and +1

(iii) Design functions

-blockboundaryPlot  This function plots a block boundary on a plot produced by 'designPlot'. It supersedes blockboundary.plot.
-designAmeasures  Calculates the A-optimality measures from the variance matrix for predictions that can be obtained using mat.Vpredicts
-designAnatomy  Given the layout for a design, obtain its anatomy via the canonical analysis of its projectors to show the confounding and aliasing inherent in the design.
-designBlocksGGPlot  Adds block boundaries to a plot produced by designGGPlot.
-designGGPlot  Plots labels on a two-way grid of coloured cells using ggplot2.
-designLatinSqrSys  Generate a systematic plan for a Latin Square design.
-designPlot  A graphical representation of an experimental design using labels stored in a matrix. It superseded design.plot.
-designPlotlabels  Plots labels on a two-way grid using ggplot2.
-designRandomize  Randomize allocated to recipient factors to produce a layout for an experiment. It supersedes fac.layout.
-designTwoPhaseAnatomies  Given the layout for a design and three structure formulae, obtain the anatomies for the two-phase, first-phase, cross-phase and second-phase designs.
-marginality.pstructure  Extracts the marginality matrix from a pstructure.object
-marginality.pstructure  Extracts a list containing the marginality matrices from a pcanonical.object
print.aliasing
print.aliasing
no.reps
no.reps
detect.diff
detect.diff
power.exp
power.exp
summary.pcanon
summary.pcanon

(iv) ANOVA functions

fitted.aovlist
fitted.aovlist
fitted.errors
fitted.errors
interaction.ABC.plot
interaction.ABC.plot
qqyeffects
qqyeffects
resid.errors
resid.errors
residuals.aovlist
residuals.aovlist
strength
strength
tukey.1df
tukey.1df
yates.effects
yates.effects

(v) Matrix functions

elements
elements
fac.ar1mat
fac.ar1mat
fac.sumop
fac.sumop
fac.vcmat
fac.vcmat
mat.I
mat.I
mat.J
mat.J
mat.ar1
mat.ar1
mat.ar2
mat.ar2
mat.ar3
mat.ar3
mat arma
mat arma
mat.arma
mat.arma
mat.arma
mat.arma
mat.arma
mat.arma
mat.arma
mat.ncssvar
mat.ncssvar
mat.sar
mat.sar
mat.sar2
mat.sar2
mat.dirprod
mat.dirprod

Prints an aliasing data.frame
Computes the number of replicates for an experiment
Computes the detectable difference for an experiment
Computes the power for an experiment
Summarizes the anatomy of a design, being the
decomposition of the sample space based on its
canonical analysis

Extract the fitted values for a fitted model
from an aovlist object
Extract the fitted values for a fitted model
Plots an interaction plot for three factors
Half or full normal plot of Yates effects
Extract the residuals for a fitted model
Extract the residuals from an aovlist object
Generate paper strength values
Performs Tukey’s
one-degree-of-freedom-test-for-nonadditivity
Extract Yates effects

Extract the elements of an array specified by
the subscripts
Forms the ar1 correlation matrix for a
(generalized) factor
Computes the summation matrix that produces
sums corresponding to a (generalized) factor
Forms the variance matrix for the variance
component of a (generalized) factor
Forms a unit matrix
Forms a square matrix of ones
Forms an ar1 correlation matrix
Forms an ar2 correlation matrix
Forms an ar3 correlation matrix
Forms an arma correlation matrix
Forms a banded matrix
Forms an exponential correlation matrix
Forms a gaussian correlation matrix
Forms an ma1 correlation matrix
Forms an ma2 correlation matrix
Forms a variance matrix for random cubic
smoothing spline effects
Forms an sar correlation matrix
Forms an sar2 correlation matrix
Forms the direct product of two matrices
**dae-package**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mat.dirsum</td>
<td>Forms the direct sum of a list of matrices</td>
</tr>
<tr>
<td>mat.Vpred</td>
<td>Forms the variance matrix of predictions</td>
</tr>
<tr>
<td>mat.Vpredicts</td>
<td>Forms the variance matrix of predictions, based on supplied matrices or formulae.</td>
</tr>
<tr>
<td>Zncsspline</td>
<td>Forms the design matrix for fitting the random effects for a natural cubic smoothing spline.</td>
</tr>
</tbody>
</table>

(vi) Projector and canonical efficiency functions

Projector class:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>projector</td>
<td>Create projectors</td>
</tr>
<tr>
<td>projector-class</td>
<td>Class projector</td>
</tr>
<tr>
<td>is.projector</td>
<td>Tests whether an object is a valid object of class projector</td>
</tr>
<tr>
<td>print.projector</td>
<td>Print projectors</td>
</tr>
<tr>
<td>correct.degree</td>
<td>Check the degrees of freedom in an object of class projector</td>
</tr>
<tr>
<td>degree</td>
<td>Degrees of freedom extraction and replacement</td>
</tr>
</tbody>
</table>

Accepts two or more formulae:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>designAnatomy</td>
<td>An anatomy of a design, obtained from a canonical analysis of the relationships between sets of projectors.</td>
</tr>
<tr>
<td>summary.pcanon</td>
<td>Summarizes the anatomy of a design, being the decomposition of the sample space based on its canonical analysis.</td>
</tr>
<tr>
<td>print.summary.pcanon</td>
<td>Prints the values in an 'summary.pcanon' object</td>
</tr>
<tr>
<td>efficiencies.pcanon</td>
<td>Extracts the canonical efficiency factors from a list of class 'pcanon'</td>
</tr>
</tbody>
</table>

Accepts exactly two formulae:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>projs.2canon</td>
<td>A canonical analysis of the relationships between two sets of projectors</td>
</tr>
<tr>
<td>projs.combine.p2canon</td>
<td>Extract, from a p2canon object, the projectors</td>
</tr>
<tr>
<td>summary.p2canon</td>
<td>A summary of the results of an analysis of the relationships between two sets of projectors</td>
</tr>
<tr>
<td>print.summary.p2canon</td>
<td>Prints the values in an 'summary.p2canon' object that give the combined decomposition</td>
</tr>
<tr>
<td>efficiencies.p2canon</td>
<td>Extracts the canonical efficiency factors from a list of class 'p2canon'</td>
</tr>
</tbody>
</table>

Accepts a single formula:
as.data.frame.pstructure Coerces a pstructure.object to a data.frame
print.pstructure Prints a pstructure.object
pstructure.formula Takes a formula and constructs a pstructure.object that includes the orthogonalized projectors for the terms in a formula

Others:
decomp.relate Examines the relationship between the eigenvectors for two decompositions
efficiency.criteria Computes efficiency criteria from a set of efficiency factors
fac.meanop Computes the projection matrix that produces means
proj2.eigen Canonical efficiency factors and eigenvectors in joint decomposition of two projectors
proj2.efficiency Computes the canonical efficiency factors for the joint decomposition of two projectors
proj2.combine Compute the projection and Residual operators for two, possibly nonorthogonal, projectors
show-methods Methods for Function ‘show’ in Package dae

(vii) Miscellaneous functions

extab Expands the values in table to a vector
get.daeTolerance Gets the value of daeTolerance for the package dae
harmonic.mean Calculates the harmonic mean.
is.allzero Tests whether all elements are approximately zero
rmvnorm Generates a vector of random values from a multivariate normal distribution
set.daeTolerance Sets the value of daeTolerance for the package dae

Author(s)
NA
Maintainer: NA

---

**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.data.frame.pstructure**

*Coerces a pstructure.object to a data.frame.*

Description

Coerces a `pstructure.object`, which is of class `pstructure`, to a `data.frame`. One can choose whether or not to include the marginality matrix in the data.frame. The aliasing component is excluded.

Usage

```r
## S3 method for class 'pstructure'
as.data.frame(x, row.names = NULL, optional = FALSE, ...,
               omit.marginality = FALSE)
```

Arguments

- `x` The `pstructure.object`, which is of class `pstructure` and is to be coerced.
- `row.names` NULL or a `character` vector giving the row names for the data frame. Missing values are not allowed.
- `optional` A `logical` passed to `as.data.frame`. If TRUE, setting row names and converting column names (to syntactic names: see `make.names`) is optional. Note that all of R's base package `as.data.frame()` methods use `optional` only for column names treatment, basically with the meaning of `data.frame(*,check.names = !optional)`.
- `...` Further arguments passed to or from other methods.
- `omit.marginality` A `logical`, which, if TRUE, results in the marginality matrix being omitted from the `data.frame`.

Value

A `data.frame` with as many rows as there are non-aliased terms in the `pstructure.object`. The columns are `df`, `terms`, `sources` and, if `omit.marginality` is FALSE, the columns of the generated `levels` with columns of the marginality matrix that is stored in the marginality component of the object.
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 a 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. If a numeric is supplied, it is left unchanged.

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**

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

---

**BIBDWheat.dat**  
*Data for a balanced incomplete block experiment*

---

**Description**

The data set comes from Joshi (1987) and is the data from an experiment to investigate six varieties of wheat that employs a balanced incomplete block design (BIBD) with ten blocks, each consisting of three plots. For more details see the vignette *daeDesignNotes.pdf*.

**Usage**

```r
data(BIBDWheat.dat)
```

**Format**

A data.frame containing 30 observations of 4 variables.

**Source**


---

**blockboundaryPlot**  
*This function plots a block boundary on a plot produced by designPlot.*

---

**Description**

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

**Usage**

```r
blockboundaryPlot(blockdefinition = NULL, blocksequence = FALSE,  
rstart= 0, cstart = 0, nrows, ncolumns,  
blocklinecolour = 1, blocklineweight = 2)
```
Arguments

blockdefinition

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.

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

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

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

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

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

blocklinecolour A character string specifying the colour of the block boundary.
See Colour specification under the par function.

blocklinewidth 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.

Author(s)

Chris Brien

See Also
designPlot, par, DiGger

Examples

## Not run:
SPL.Lines.mat <- matrix(as.numfac(Lines), ncol=16, byrow=T)
colnames(SPL.Lines.mat) <- 1:16ownames(SPL.Lines.mat) <- 1:10
SPL.Lines.mat <- SPL.Lines.mat[10:1, 1:16]
A design for one of the growth cabinets in an experiment with 50 lines and 4 harvests

Description

The systematic design for a lattice square for 49 treatments consisting of four 7 x 7 squares. For more details see the vignette daeDesignNotes.pdf.

Usage

data(Cabinet1.des)

Format

A data.frame containing 160 observations of 15 variables.
Casuarina.dat  
Data for an experiment with rows and columns from Williams (2002)

Description

Williams (2002, p.144) provides an example of a resolved, Latinized, row-column design with four rectangles (blocks) each of six rows by ten columns. The experiment investigated differences between 60 provenances of a species of Casuarina tree, these provenances coming from 18 countries; the trees were inoculated prior to planting at two different times, time of inoculation being assigned to the four replicates so that each occurred in two replicates. At 30 months, diameter at breast height (Dbh) was measured. For more details see the vignette daeDesignNotes.pdf.

Usage

data(Casuarina.dat)

Format

A data.frame containing 240 observations of 7 variables.

Source


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 Machine$double.eps ^ 0.5 (about 1.5E-08). 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

```r
## 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)
```

---

dae-deprecated  Deprecated Functions in Package dae

Description

These functions have been renamed and deprecated in dae.

Usage

Ameasures(...)  blockboundary.plot(...)
design.plot(...)  proj2.decomp(...)
proj2.ops(...)  projs.canon(...)
projs.structure(...)  

Arguments

... absorbs arguments passed from the old functions of the style foo.bar().
Author(s)
Chris Brien

daeTips
The intermittent, randomly-presented, startup tips.

Description
The intermittent, randomly-presented, startup tips.

Startup tips
Need help? Enter help(package = 'dae') and click on 'User guides, package vignettes and other docs'.
Need help? The manual is in the doc subdirectory of the package's install directory.
Find out what has changed in dae: enter news(package = 'dae').
Need help to produce randomized designs? Enter vignette('DesignNotes', package = 'dae').
Need help to do the canonical analysis of a design? Enter vignette('DesignNotes', package = 'dae').
Use suppressPackageStartupMessages() to eliminate all package startup messages.
To see all the intermittent, randomly-presented, startup tips enter ?daeTips.
For versions between CRAN releases (and more) go to http://chris.brien.name/rpackages.

Author(s)
Chris Brien

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

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

Usage
decomp.relate(decomp1, decomp2)
Arguments

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

Details

Each element of the $r_1 \times r_2$ 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 .Machine$double.eps ^ 0.5 (about 1.5E-08). The function set.daeTolerance can be used to change daeTolerance.

Value

A matrix that is $r_1 \times r_2$ where $r_1$ and $r_2$ 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.eigen, proj2.combine in package dae, eigen.

Examples

```r
## PBIBD(2) from p. 379 of Cochran and Cox (1957) Experimental Designs.
## 2nd edn Wiley, New York
PBIBD2.unit <- list(Block = 6, Unit = 4)
PBI0D2.nest <- list(Unit = "Block")
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))
PBI0D2.lay <- designRandomize(allocated = trt,
                         recipient = PBIBD2.unit,
                         nested.recipients = PBIBD2.nest)

##obtain sets of projectors
unit.struct <- pstructure(~ Block/Unit, data = PBIBD2.lay)
trt.struct <- pstructure(~ trt, data = PBIBD2.lay)

## obtain intra- and inter-block decompositions
decomp.inter <- proj2.eigen(unit.struct$Q["Block"], trt.struct$Q["trt"])
decomp.intra <- proj2.eigen(unit.struct$Q["Unit[Block]"], trt.struct$Q["trt"])

## 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 calculated.

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 .Machine$double.eps ^ 0.5 (about 1.5E-08). The function set.daeTolerance can be used to change daeTolerance.

Value

An object of class "projector" that consists of a square, symmetric, 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**

```r
## 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
degree(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
degree(proj.m) <- proj.m
print(proj.m)
```

**designAmeasures**

*Calculates the average variance of pairwise differences from the variance matrix for predictions that can be obtained using mat.Vpredicts*

**Description**

Calculates the average variance of pairwise differences between (of elementary contrasts of) predictions, possibly for different subgroups of the predictions, from the variance matrix for the predictions. If groups are specified then the A-optimality measures are calculated for the differences between predictions within each group and for those between predictions from different groups. If groupsizes are specified, but groups are not, the predictions will be sequentially broken into groups of the size specified by the elements of groupsizes. The groups can be named.

**Usage**

```r
designAmeasures(Vpred, groupsizes = NULL, groups = NULL)
```

**Arguments**

- **Vpred**
  - The variance matrix of the predictions.
- **groupsizes**
  - A numeric containing group sizes. The sum of the elements of groupsizes must be less than or equal to the order of Vpred. If groupsizes is a named vector, the names are used to label the groups. If NULL, either groups is used or the average for all pairwise differences is obtained.
- **groups**
  - A list, each element of which specifies a subgroup of the predictions over whose pairwise differences the variances are to be averaged. If there is more than one group, the variances of all between and within group pairwise differences are averaged. If the elements of groups are named, the names are used to label the groups. If groups is NULL, either groupsizes is used or the average for all pairwise differences is obtained.
Value

A matrix containing the within and between group A-optimality measures.

Author(s)

Chris Brien

References


See Also

mat.Vpred, designAnatomy.

Examples

```r
## Reduced example from Smith et al. (2015)
## Generate two-phase design
mill.fac <- fac.gen(list(Mrep = 2, Mday = 2, Mord = 3))
field.lay <- fac.gen(list(Frep = 2, Fplot = 4))
field.lay$Variety <- factor(c("D","E","Y","W","G","D","E","M"),
  levels = c("Y","W","G","M","D","E"))
start.design <- cbind(mill.fac, field.lay[c(3,4,5,8,1,7,3,4,5,8,6,2),])
rownames(start.design) <- NULL

## Set up matrices
n <- nrow(start.design)
W <- model.matrix(~ -1+ Variety, start.design)
ng <- ncol(W)
Gg<- diag(1, ng)
Vu <- with(start.design, fac.vcmat(Mrep, 0.3) +
  fac.vcmat(fac.combine(list(Mrep, Mday)), 0.2) +
  fac.vcmat(Frep, 0.1) +
  fac.vcmat(fac.combine(list(Frep, Fplot)), 0.2))
R <- diag(1, n)

## Calculate the variance matrix of the predicted random Variety effects
Vp <- mat.Vpred(W = W, Gg = Gg, Vu = Vu, R = R)

designAmeasures(Vp)
designAmeasures(Vp, groups=list(fldUndup = c(1:4), fldDup = c(5,6)))
grpsizes <- c(4,2)
names(grpsizes) <- c("fldUndup", "fldDup")
designAmeasures(Vp, groupsizes = grpsizes)
designAmeasures(Vp, groupsizes = c(4))
designAmeasures(Vp, groups=list(c(1,4),c(5,6)))

designAmeasures(Vp, groups=list(c(1,4),c(5,6)))

## Calculate the variance matrix of the predicted fixed Variety effects, eliminating the grand mean
```
designAnatomy

Given the layout for a design, obtain its anatomy via the canonical analysis of its projectors to show the confounding and aliasing inherent in the design.

Description

Computes the canonical efficiency factors for the joint decomposition of two or more structures or sets of mutually orthogonally projectors (Brien and Bailey, 2009; Brien, 2017; Brien, 2019), orthogonalizing projectors in a set to those earlier in the set of projectors with which they are partially aliased. The results can be summarized in the form of a decomposition table that shows the confounding between sources from different sets. For examples of its use also see the vignette daeDesignNotes.pdf.

Usage

designAnatomy(formulae, data, keep.order = TRUE, grandMean = FALSE, orthogonalize = "hybrid", labels = "sources", marginality = NULL, check.marginality = TRUE, which.criteria = c("aefficiency","eefficiency","order"), aliasing.print = FALSE, omit.projectors = c("pcanon", "combined"), ...)

Arguments

formulae An object of class list whose components are of class formula. Usually, the terms in a single formula have the same status in the allocation of factors in the design. For example, all involve only factors that were allocated, or all involve factors that were recipients of allocated factors. The names of the components are used to identify the sources in the summary.pcanon object. They will also be used to name the terms, sources and marginality lists in the pcanon.object.

data A data.frame contains the values of the factors and variables that occur in formulae.

keep.order A logical indicating whether the terms should keep their position in the expanded formula projector, or reordered so that main effects precede two-factor interactions, which precede three-factor interactions and so on.

grandMean A logical indicating whether the projector for the grand mean is to be included for each structure.
orthogonalize

A character vector indicating the method for orthogonalizing a projector to those for terms that occurred previously in a single formula. Three options are available: hybrid; differencing; eigenmethods. The hybrid option is the most general and uses the relationships between the projection operators for the terms in the formula to decide which projectors to subtract and which to orthogonalize using eigenmethods. The differencing option subtracts, from the current projector, those previously orthogonalized projectors for terms whose factors are a subset of the current projector’s factors. The eigenmethods option recursively orthogonalizes the projects using an eigenanalysis of each projector with previously orthogonalized projectors. If a single value is given, it is used for all formulae.

labels

A character nominating the type of labels to be used in labelling the projectors, and which will be used also in the output tables, such the tables of the aliasing in the structure. The two alternatives are terms and sources. Terms have all factors/variables in it separated by colons (:) . Sources have factors/variables in them that represent interactions separated by hashes (#); if some factors are nested within others, the nesting factors are surrounded by square brackets ([ and ]) and separated by colons (:) . If some generalized, or combined, factors have no marginal terms, the constituent factors are separated by colons (:) and if they interact with other factors in the source they will be parenthesized.

marginality

A list that can be used to supply some or all of the marginality matrices when it is desired to overwrite calculated marginality matrices or when they are not calculated. If the list is the same length as the formulae list, they will be associated in parallel with the components of formulae, irrespective of the naming of the two lists. If the number of components in marginality is less than the number of components in formulae then both lists must be named so that those in the marginality list can be matched with those in the formulae list.

Each component of the marginality list must be either NULL or a square matrix consisting of zeroes and ones that gives the marginalites of the terms in the formula. It must have the row and column names set to the terms from the expanded formula, including being in the same order as these terms. The entry in the i-th row and j-th column will be one if the i-th term is marginal to the j-th term i.e. the column space of the i-th term is a subspace of that for the j-th term and so the source for the j-th term is to be made orthogonal to that for the i-th term. Otherwise, the entries are zero. A row and column should not be included for the grand mean even if grandMean is TRUE.

check.marginality

A logical indicating whether the marginality matrix, when it is supplied, is to be checked against that computed by pstructure.formula. It is ignored when orthogonalize is set to eigenmethods.

which.criteria

A character vector nominating the efficiency criteria to be included in the summary of aliasing between terms within a structure. It can be none, all or some combination of aefficiency, mefficiency, sefficiency, eefficiency, xefficiency, order and dforthog – for details see efficiency.criteria. If none, no summary is printed.

aliasing.print

A logical indicating whether the aliasing between sources is to be printed.
omit.projectors

A character vector of the types of projectors to omit from the returned pcanon object. If pcanon is included in the vector then the projectors in these objects will be replaced with a numeric containing their degrees of freedom. If combined is included in the vector then the projectors for the combined decomposition will be replaced with a numeric containing their degrees of freedom. If none is included in the vector then no projectors will be omitted.

... further arguments passed to terms.

Details

For each formula supplied in formulae, the set of projectors is obtained using pstructure; there is one projector for each term in a formula. Then projs.2canon is used to perform an analysis of the canonical relationships between two sets of projectors for the first two formulae. If there are further formulae, the relationships between its projectors and the already established decomposition is obtained using projs.2canon. The core of the analysis is the determination of eigenvalues of the product of pairs of projectors using the results of James and Wilkinson (1971). However, if the order of balance between two projection matrices is 10 or more or the James and Wilkinson (1971) methods fails to produce an idempotent matrix, equation 5.3 of Payne and Tobias (1992) is used to obtain the projection matrices for their joint decomposition.

Value

A pcanon.object.

Author(s)

Chris Brien

References


See Also
designRandomize, designLatinSqrSys, designPlot, pcanon.object, p2canon.object, summary.pcanon, efficiencies.pcanon, pstructure, projs.2canon, proj2.efficiency, proj2.combine, proj2.eigen, efficiency.criteria, 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(Block = 6, Unit = 4)
PBIBD2.nest <- list(Unit = "Block")
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 <- designRandomize(allocated = trt,
recipient = PBIBD2.unit,
nested.recipients = PBIBD2.nest)

##obtain combined decomposition and summarize
unit.trt.canon <- designAnatomy(formulae = list(unit=~ Block/Unit, trt=~ trt),
data = PBIBD2.lay)
summary(unit.trt.canon, which.criteria = c("aeff","eeff","order"))
summary(unit.trt.canon, which.criteria = c("aeff","eeff","order"), labels.swap = TRUE)

## Three-phase sensory example from Brien and Payne (1999)
## Not run:
data(Sensory3Phase.dat)
Eval.Field.Treat.canon <- designAnatomy(formulae = list( eval=~ ((Occasions/Intervals/Sittings)*Judges)/Positions, field=~ (Rows*(Squares/Columns))/Halfplots, treats=~ Trellis*Method),
data = Sensory3Phase.dat)
summary(Eval.Field.Treat.canon, which.criteria =c("aefficiency", "order"))

## End(Not run)

designBlocksGGPlot  Adds block boundaries to a plot produced by designGGPlot.

Description

This function adds block boundaries to a plot produced by designGGPlot. It allows control of the starting unit, through originrow and origincolumn, and the number of rows (nrows) and columns (ncolumns) from the starting unit that the blocks to be plotted are to cover.

Usage

designBlocksGGPlot(ggplot.obj, blockdefinition = NULL, blocksequence = FALSE,
originrow= 0, origincolumn = 0, nrows, ncolumns, blocklinecolour = "blue", blocklinesize = 2,
printPlot = TRUE)
**Arguments**

- **ggplot.obj**
  An object produced by `ggplot`.

- **blockdefinition**
  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.

- **blocksequence**
  A logical that determines whether block numbers are repetitions or sequences of block numbers.

- **originrow**
  A numeric specifying the row after which the plotting of block boundaries is to start.

- **origincolumn**
  A numeric specifying the column after which the plotting of block boundaries is to start.

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

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

- **blocklinecolour**
  A character string specifying the colour of the block boundary. See Colour specification under the `par` function.

- **blocklinesize**
  A numeric giving the width of the block boundary to be plotted.

- **printPlot**
  A logical indicating whether to print the plot after adding the block boundaries.

**Value**

An object of class "ggplot", formed by adding to the input `ggplot.obj` and which can be plotted using `print`.

**Author(s)**

Chris Brien

**Source**

See Also

designGGPlot.par, DiGGer

Examples

## Construct a randomized layout for the split-unit design described by
## Brien et al. (2011, Section 5)

```r
split.sys <- cbind(fac.gen(list(Months = 4, Athletes = 3, Tests = 3)),
                    fac.gen(list(Intensities = LETTERS[1:3], Surfaces = 3),
                            times = 4))

split.lay <- designRandomize(allocated = split.sys[, c("Intensities", "Surfaces")],
                              recipient = split.sys[, c("Months", "Athletes", "Tests")],
                              nested.recipients = list(Athletes = "Months",
                                                      Tests = c("Months", "Athletes")),
                              seed = 2598)

## Plot the design

cell.colours <- c("lightblue", "lightcoral", "lightgoldenrod", "lightgreen", "lightgrey",
                  "lightpink", "lightsalmon", "lightcyan", "lightyellow", "lightseagreen")

split.lay <- within(split.lay,
                    Treatments <- fac.combine(list(Intensities, Surfaces),
                                               combine.levels = TRUE))

plt <- designGGPlot(split.lay, labels = "Treatments",
                    row.factors = "Tests", column.factors = c("Months", "Athletes"),
                    colour.values = cell.colours[1:9], size = 6,
                    blockdefinition = rbind(c(3, 1)), blocklinecolour = "darkgreen",
                    printPlot = FALSE)

# Add Month boundaries

designBlocksGGPlot(plt, nrows = 3, ncolumns = 3, blockdefinition = rbind(c(3, 3)))

#### A layout for a growth cabinet experiment that allows for edge effects

data(Cabinet1.des)

plt <- designGGPlot(Cabinet1.des, labels = "Combinations", cellalpha = 0.75,
                     title = "Lines and Harvests allocation for Cabinet 1",
                     printPlot = FALSE)

## Plot Mainplot boundaries

plt <- designBlocksGGPlot(plt, blockdefinition = cbind(4, 16), originrow = 1,
                          blocklinecolour = "green", nrows = 9, ncolumns = 16,
                          printPlot = FALSE)

plt <- designBlocksGGPlot(plt, blockdefinition = cbind(1, 4),
                          blocklinecolour = "green", nrows = 1, ncolumns = 16,
                          printPlot = FALSE)

plt <- designBlocksGGPlot(plt, blockdefinition = cbind(1, 4), originrow = 9,
                          blocklinecolour = "green", nrows = 10, ncolumns = 16,
                          printPlot = FALSE)

## Plot all 4 block boundaries

plt <- designBlocksGGPlot(plt, blockdefinition = cbind(8, 5, 4), blocksequence = TRUE,
                          origincolumn = 1, originrow = 1,
                          blocklinecolour = "blue", nrows = 9, ncolumns = 15,
                          printPlot = FALSE)
```
designGGPlot

Plots labels on two-way grids of coloured cells using ggplot2

Description

Plots the labels in a grid of cells specified by row.factors and column.factors. The cells can be coloured by the values of the column specified by column.name and can be divided into facets by specifying multiple row and or column factors.

Usage

designGGPlot(design, labels = NULL,
row.factors = "Rows", column.factors = "Columns",
cellfillcolour.column = NULL, colour.values = NULL,
cellalpha = 1, celllinetype = "solid", celllinesize = 0.5,
celllinecolour = "black", cellheight = 1, cellwidth = 1,
reverse.x = FALSE, reverse.y = TRUE,
xlab, ylab, title, title.size = 15, axis.text.size = 15,
blocksequence = FALSE, blockdefinition = NULL,
blocklinewidth = 2, blocklinecolour = "blue", blocklinesize = 2,
printPlot = TRUE, ggplotFuncs = NULL, ...)

Arguments

design A data.frame containing labels, column.factors, row.factors and, if specified, colour.column.

labels A character giving the name of the column in data containing the labels to be plotted on the grid. If labels is NULL, no labels are added.

row.factors A character giving the names of the factors (or numerics) in data that index the rows of the plot grid used to represent the design. If there is more than one name, then facet_grid is used to facet the plot in the y direction, based on all but the last name. The factor corresponding to the last name will index the rows in each facet.

column.factors A character giving the names of the factors (or numerics) in data that index the columns of the plot grid used to represent the design. If there is more than one name, then facet_grid is used to facet the plot in the x direction, based
on all but the last name. The factor corresponding to the last name will index
the columns in each facet.

reverse.x A logical which, if true, causes the order of values on the x-axis to be reversed,
the natural order being to increase from left to right.

reverse.y A logical which, if true, causes the order of values on the y-axis to be reversed,
the natural order being to increase from bottom to top.

cellfillcolour.column A character giving the name of the column in data that is to be used to vary
the colour the used to fill a cell.

colour.values A character giving the name or names of the colours to be used in filling the
cell. If cellfillcolour.column is not NULL then the number of colours specified
needs to match the number of unique values in the cellfillcolour.column.

cellalpha A numeric specifying the degree of transparency to be used in cell fill. It is
a ratio in which the denominator specifies the number of points (or lines) that
must be overplotted to give a solid cover.

celllinetype A numeric or character giving the type of line for the cell border. An integer
or name: 0 = blank, 1 = solid, 2 = dashed, 3 = dotted, 4 = dotdash, 5 = longdash,
6 = twodash. For more information see vignette("ggplot2-specs").

celllinesize A numeric specifying the size of the line in mm.

celllinecolour A character giving the name of the colour to use for the cell outline.

cellheight A numeric specifying the height of a cell.

cellwidth A numeric specifying the width of a cell.

xlab Label for the x-axis. By default it is the last name in the column.factors.

ylab Label for the y-axis. By default it is the last name in the row.factors.

title Title for plot window. By default it is "Plot of labels".

title.size A numeric giving the size for all titles: xlab, ylab and title.

axis.text.size A numeric giving the size for tick labels.

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

blockdefinition 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.

blocklinecolour A character string specifying the colour of the block boundary.
See also the scale_colour_* functions or Colour specification under the
par function.
**designGGPlot**

**blocklinesize**  A numeric giving the width of the block boundary to be plotted.

**printPlot**  A logical indicating whether to print the plot produced.

**ggplotFuncs**  A list, each element of which contains the results of evaluating a ggplot function. It is created by calling the list function with a ggplot function call for each element.

... Other arguments that are passed down to the geom_text call that plots the labels.

**Value**

An object of class "ggplot", which can be plotted using print.

**Author(s)**

Chris Brien

**See Also**

fac.combine in package dae, designPlot.

**Examples**

#### Plot a randomized complete block design

```r
treatments <- factor(rep(1:6, times = 5))
rcbd.lay <- designRandomize(allocated = treatments,
                           recipient = list(Blocks = 5, Units = 6),
                           nested.recipients = list(Units = "Blocks"),
                           seed = 74111)
designGGPlot(rcbd.lay, labels = "Treatments", size = 5,
             row.factors = "Blocks", column.factors = "Units",
             blockdefinition = cbind(1,5))
```

#### Plot without labels

```r
designGGPlot(rcbd.lay, cellfillcolour.column = "Treatments",
             row.factors = "Blocks", column.factors = "Units",
             colour.values = c("lightblue","lightcoral","lightgoldenrod",
                               "lightgreen","lightgrey", "lightpink"),
             blockdefinition = cbind(1,6))
```

#### Plot a lattice square design

```r
data(LatticeSquare_t49.des)
designGGPlot(LatticeSquare_t49.des, labels = "Lines", size = 5,
             row.factors = c("Intervals", "Runs"), column.factors = "Times",
             blockdefinition = cbind(7,7))
```
Generate a systematic plan for a Latin Square design

Description
Generates a systematic plan for a Latin Square design using the method of cycling the integers 1 to the number of treatments. The start of the cycle for each row, or the first column, can be specified as a vector of integers.

Usage
designLatinSqrSys(order, start = NULL)

Arguments
order The number of treatments.
start A numeric containing order unique values between one and order. These are interpreted as the value for the first column for each row. If NULL, 1:order is used.

Value
A numeric containing order x order integers between 1 and order such that, when the numeric is considered as a square matrix of size order, each integer occurs once and only once in each row and column of the matrix.

See Also
designRandomize, designPlot, designAnatomy in package dae.

Examples
matrix(designLatinSqrSys(5, start = c(seq(1, 5, 2), seq(2, 5, 2))), nrow=5)
designLatinSqrSys(3)

designPlot
A graphical representation of an experimental design using labels stored in a matrix.

Description
This function uses labels, usually derived from treatment and blocking factors from an experimental design and stored in a matrix, to build a graphical representation of the matrix, highlighting the position of certain labels. 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

designPlot(designMatrix, labels = NULL, altlabels = NULL, plotlabels = TRUE, 
rttitle = NULL, ctitle = NULL, 
rlabelsreverse = FALSE, clabelsreverse = FALSE, 
font = 1, chardivisor = 2, rchardivisor = 1, cchardivisor = 1, 
cellfillcolour = NA, plotcellboundary = TRUE, 
rcellpropn = 1, ccellpropn = 1, 
blocksequence = FALSE, blockdefinition = NULL, 
blocklineweight = 2, 
rotate = FALSE, new = TRUE, ...)

Arguments

designMatrix      A matrix containing a set of numerics or characters being the labels as they have been assigned to the cells of the grid represented by the matrix.

labels            A numeric or character vector giving the cells in designMatrix that are to be plotted in this call to designPlot. If NULL then all the cells are plotted. What is actually plotted for a cell is controlled jointly by labels, plotlabels, altlabels, plotcellboundary and cellfillcolour. If plotlabels is TRUE and altlabels is NULL then labels are plotted on the cells, unless labels is NULL when the labels in designMatrix are plotted. Whatever is being plotted, altlabels and cellfillcolour must have an appropriate number of values. See text for more information on specifying the labels.

altlabels         Either a character vector containing an alternative set of labels for the labels currently being plotted or a single integer specifying an alternative symbol to be used in plotting cells when plotlabels is TRUE. The length of altlabels must be one or the same length as labels, unless labels is NULL in which case it must equal the number of unique labels in designMatrix. If altlabels is NULL, the labels specified in labels are plotted when plotlabels is TRUE. If labels is also NULL, the labels in designMatrix are plotted. See text for more information on specifying the labels.

plotlabels        A logical to indicate whether labels are to be plotted in the cells. If TRUE, print all labels or the specific labels listed in labels. If FALSE, no labels are printed in the cells.

rttitle           A character string to use as a title for rows of the plot. If rttitle is NULL then no title is plotted.

ctitle            A character string to use as a title for columns of the plot. If ctitle is NULL then no title is plotted.

rlabelsreverse    A logical indicating whether to reverse the row labels.

clabelsreverse    A logical indicating whether to reverse the column labels.

font              An integer specifying the font to be used for row and column labelling. See par for further details.

chardivisor       A numeric that changes the size of text and symbols in the cells by dividing the default size by it.
rchardivisor  A numeric that changes the size of the labels of the rows of the design by dividing the default size by it.

cchardivisor  A numeric that changes the size of the labels of the columns of the design by dividing the default size by it.

cellfillcolour  A character string specifying the colour of the fill for the cells to be plotted in this call. If there is only one colour then all cells being plotted with that colour. If there is more than one colour then, unless labels is NULL, the number of colours must at least equal the number of labels and then the fill colours will be matched, one for one from the first colour, with the labels. If labels is NULL then the number of colours must at least equal the number of unique labels in designMatrix. The default, NA, is to leave the cells unfilled.

See also Colour specification under the par function.

plotcellboundary  A logical indicating whether a boundary is to be plotted around a cell.

cellpropn  a value between 0 and 1 giving the proportion of the standard row size of a cell size to be plotted as a cell.

cellpropn  a value between 0 and 1 giving the proportion of the standard column size of a cell size to be plotted as a cell.

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

blockdefinition  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.

blocklinecolour  A character string specifying the colour of the block boundary. See also Colour specification under the par function.

blocklinewidth  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.

... further arguments passed to polygon in plotting the cell.

Value

no values are returned, but a plot is produced.
**designPlotlabels**

Plots labels on a two-way grid using ggplot2

**Description**

Plots the labels in a grid specified by `grid.x` and `grid.y`. The labels can be coloured by the values of the column specified by `column.name`.

**Usage**

```r
designPlotlabels(data, labels, grid.x = "Columns", grid.y = "Rows",
                 colour.column=NULL, colour.values=NULL,
                 reverse.x = FALSE, reverse.y = TRUE,
                 xlab, ylab, title, printPlot = TRUE, ggplotFuncs = NULL, ...)
```
**Arguments**

- **data**
  A `data.frame` containing labels, grid.x, grid.y and, if specified, colour.column.
- **labels**
  A character giving the name of the column in data containing the labels to be plotted on the grid.
- **grid.x**
  A character giving the name of the column in data that specifies the x-coordinates of the plot grid.
- **grid.y**
  A character giving the name of the column in data that specifies the y-coordinates of the plot grid.
- **reverse.x**
  A logical which, if true, causes the order of values on the x-axis to be reversed.
- **reverse.y**
  A logical which, if true, causes the order of values on the y-axis to be reversed.
- **colour.column**
  A character giving the name of the column in data that is to be used to colour the values plotted on the grid.
- **colour.values**
  A character giving the name of the column in data that is to be used to colour the values plotted on the grid.
- **xlab**
  Label for the x-axis. By default it is the name of the grid.x.
- **ylab**
  Label for the y-axis. By default it is the name of the grid.y.
- **title**
  Title for plot window. By default it is "Plot of labels".
- **printPlot**
  A logical indicating whether to print the plot.
- **ggplotFuncs**
  A list, each element of which contains the results of evaluating a `ggplot` function. It is created by calling the `list` function with a `ggplot` function call for each element.

**Value**

An object of class "ggplot", which can be plotted using `print`.

**Author(s)**

Chris Brien

**See Also**

`fac.combine` in package `dae`, `designPlot`.

**Examples**

```r
Treatments <- factor(rep(1:6, times = 5))
RCBD.lay <- designRandomize(allocated = Treatments,
                          recipient = list(Blocks = 5, Units = 6),
                          nested.recipients = list(Units = "Blocks"),
                          seed = 74111)
designPlotlabels(RCBD.lay, labels = "Treatments",
                 grid.x = "Units", grid.y = "Blocks",
                 colour.column = "Treatments", size = 5)
```
A systematic design is specified by a set of allocated factors that have been assigned to a set of recipient factors. In textbook designs the allocated factors are the treatment factors and the recipient factors are the factors indexing the units. To obtain a randomized layout for a systematic design it is necessary to provide (i) the systematic arrangement of the allocated factors, (ii) a list of the recipient factors or a data.frame with their values, and (iii) the nesting of the recipient factors for the design being randomized. Given this information, the allocated factors will be randomized to the recipient factors, taking into account the nesting between the recipient factors for the design. However, allocated factors that have different values associated with those recipient factors that are in the except vector will remain unchanged from the systematic design.

Also, if allocated is NULL then a random permutation of the recipient factors is produced that is consistent with their nesting as specified by nested.recipients.

For examples of its use also see the vignette daeDesignNotes.pdf.

Usage

designRandomize(allocated = NULL, recipient, nested.recipients = NULL, except = NULL, seed = NULL, unit.permutation = FALSE, ...)

Arguments

allocated A factor or a data.frame containing the systematically allocated values of the factor(s). If NULL, a random permutation of the recipient factors is produced that is consistent with their nesting as specified by nested.recipients.

recipient A data.frame or a list of factors, along with their levels that specify the set of recipient factors that are allocated levels of the allocated factors. If a list, the name of each component of the list is a factor name and the component is either (i) a single numeric value that is the number of levels, (ii) a numeric vector that contains the levels of the factor, (iii) or a character vector that contains the labels of the levels of the factor. The values of factors will be generated in standard order using fac.gen and so the values in allocated must match this.

nested.recipients A list of the recipient factors that are nested in other factors in recipient. 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).
A character vector containing the names of recipient factors that are to be excepted from the permutation; any allocated factors whose values differ between the levels of the factors in this vector will not have those values randomized.

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

A logical indicating whether to include the .Unit and .Permutation columns in the data.frame.

Further arguments passed to or from other methods. Unused at present.

Details

A systematic design is specified by the matching of the supplied allocated and recipient factors. If recipient is a list then fac.gen is used to generate a data.frame with the combinations of the levels of the recipient factors in standard order. Although, the data.frames are not combined at this stage, the systematic design is the combination, by columns, of the values of the allocated factors with the values of recipient factors in the recipient data.frame.

The method of randomization described by Bailey (1981) is used to randomize the allocated factors to the recipient factors. That is, a permutation of the recipient factors is obtained that respects the nesting for the design, but does not permute any of the factors in the except vector. This permutation is applied to the recipient factors. Then the data.frame containing the permuted recipient factors and that containing the unpermuted allocated factors are combined columnwise, as in cbind. To produce the randomized layout, the rows of the combined data.frame are reordered so that its recipient factors are in either standard order or, if a data.frame was supplied to recipient, the same order as for the supplied data.frame.

The .Units and .Permutation vectors enable one to swap between this permutation and the randomized layout. The $i$th value in .Permutation gives the unit to which unit $i$ was assigned in the randomization.

Value

A data.frame with the values for the recipient and allocated factors that specify the layout for the experiment and, if unit.permutation is TRUE, the values for .Units and .Permutation vectors.

Author(s)

Chris Brien

References


See Also

fac.gen, designLatinSqrSys, designPlot, designAnatomy in package dae.
Examples

```r
## Generate a randomized layout for a 4 x 4 Latin square
## (the nested.recipients argument is not needed here as none of the
## factors are nested)
## Firstly, generate a systematic layout
LS.sys <- cbind(fac.gen(list(row = c("I","II","III","IV"),
    col = c(0,2,4,6))),
    treat = factor(designLatinSqrSys(4), label = LETTERS[1:4]))
## obtain randomized layout
LS.lay <- designRandomize(allocated = LS.sys["treat"],
    recipient = LS.sys[c("row","col")],
    seed = 7197132, unit.permutation = TRUE)

## 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
## Firstly, generate a systematic order such that levels of the
## treatment factor coincide with plot
RCBD.sys <- cbind(fac.gen(list(rep = 2, plot=1:3, block = c("I","II"))),
    tr = factor(rep(1:3, each=2, times=2)))
## obtain randomized layout
RCBD.lay <- designRandomize(allocated = RCBD.sys["tr"],
    recipient = RCBD.sys[c("rep","block","plot")],
    nested.recipients = list(plot = c("block","rep"),
        block="rep"),
    seed = 9719532,
    unit.permutation = TRUE)
#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.
## Firstly, generate a systematic layout
SPL.sys <- cbind(fac.gen(list(block = 2, main.unit = 4, split.unit = 3)),
    fac.gen(list(A = 4, B = 3), times = 2))
## obtain randomized layout
SPL.lay <- designRandomize(allocated = SPL.sys[c("A","B")],
    recipient = SPL.sys[c("block", "main.unit", "split.unit")],
    nested.recipients = list(main.unit = "block",
        split.unit = c("block", "main.unit")),
    seed=155251978)

## Generate a permutation of Seedlings within Species
seed.permute <- designRandomize(recipient = list(Species = 3, Seedlings = 4),
    nested.recipients = list(Seedlings = "Species"),
    seed = 75724, except = "Species",
    unit.permutation = TRUE)
```
designTwophaseAnatomies

Given the layout for a design and three structure formulae, obtain the anatomies for the two-phase, first-phase, cross-phase and second-phase designs.

Description

Uses designAnatomy to obtain the four species of designs associated with a two-phase design that are described by Brien (2019): the anatomies for the two-phase, first-phase, cross-phase and second-phase designs.

Usage

designTwophaseAnatomies(formulae, data, which.designs = "all", printAnatomies = TRUE, orthogonalize = "hybrid", marginality = NULL, which.criteria = c("aefficiency", "eefficiency", "order"), ...)

Arguments

formulae An object of class list with three components of class formula. Usually, the terms in a single formula have the same status in the allocation of factors in the design. For example, all involve only factors that were allocated, or all involve factors that were recipients of allocated factors. The names of the components are used to identify the sources in the summary.pcanon object. They will also be used to name the terms, sources and marginality lists in the pcanon.object.

data A data.frame contains the values of the factors and variables that occur in formulae.

which.designs A character vector indicating the species of designs that are to be obtained. It should include one or more of two-phase, first-phase, cross-phase and second-phase; all, the default, results in all four being obtained.

printAnatomies A logical indicating whether or not the anatomies are to be printed.

orthogonalize A character vector indicating the method for orthogonalizing a projector to those for terms that occurred previously in a single formula. Three options are available: hybrid; differencing; eigenmethods. The hybrid option is the most general and uses the relationships between the projection operators for the terms in the formula to decide which projectors to subtract and which to orthogonalize using eigenmethods. The differencing option subtracts, from the current projector, those previously orthogonalized projectors for terms whose factors are a subset of the current projector’s factors. The eigenmethods option recursively orthogonalizes the projects using an eigenanalysis of each projector with previously orthogonalized projectors. If a single value is given, it is used for all formulae.
A list that can be used to overwrite calculated marginality matrices or when they are not calculated. If the list is the same length as the formulae list, they will be associated in parallel with the components of formulae, irrespective of the naming of the two lists. If the number of components in marginality is less than the number of components in formulae then both lists must be named so that those in the marginality list can be matched with those in the formulae list.

Each component of the marginality list must be either NULL or a square matrix consisting of zeroes and ones that gives the marginalities of the terms in the formula. It must have the row and column names set to the terms from the expanded formula, including being in the same order as these terms. The entry in the ith row and jth column will be one if the ith term is marginal to the jth term i.e. the column space of the ith term is a subspace of that for the jth term and so the source for the jth term is to be made orthogonal to that for the ith term. Otherwise, the entries are zero. A row and column should not be included for the grand mean even if grandMean is TRUE.

A character vector nominating the efficiency criteria to be included in the summary of aliasing between terms within a structure. It can be none, all or some combination of aefficiency, mefficiency, sefficiency, eefficiency, xefficiency, order and dforthog – for details see efficiency.criteria. If none, no summary is printed.

Further arguments passed to designAnatomy.

Details

To produce the anatomies, designAnatomy is called. The two-phase anatomy is based on the three formulae supplied in formulae, the first-phase anatomy uses the second and third formulae, the cross-phase anatomy derives from the first and third formulae and the second-phase anatomy is obtained with the first and second formulae.

Value

A list containing the components twophase, first, cross and second. Each contains the pcanon.object for one of the four designs produced by designTwophaseAnatomies, unless it is NULL because the design was omitted from the which.designs argument.

Author(s)

Chris Brien

References


See Also

designAnatomy, pcanon.object, p2canon.object, summary.pcanon, efficiencies.pcanon, pstructure, projs.2canon, proj2.efficiency, proj2.combine, proj2.eigen, efficiency.criteria, in package dae, eigen.

projector for further information about this class.

Examples

```r
## Microarray example from Jarrett & Ruggiero (2008) - see Brien (2019)
jr.lay <- fac.gen(list(Set = 7, Dye = 2, Array = 3))
jr.lay <- within(jr.lay,
  {
    Block <- factor(rep(1:7, each=6))
    Plant <- factor(rep(c(1,2,3,2,3,1), times=7))
    Sample <- factor(c(rep(c(2,1,2,1,1,2,1,1,2,2,1,1), times=3),
      2,1,2,1,1))
    Treat <- factor(c(1,2,4,2,4,1,2,3,5,3,5,2,3,4,6,4,6,3,
      4,5,7,5,7,4,5,6,1,6,1,5,6,7,2,7,2,6,
      7,1,3,1,3,7),
      labels=c("A","B","C","D","E","F","G"))
  })

jr.anat <- designTwophaseAnatomies(formulae = list(array = ~ (Set:Array)*Dye,
  plot = ~ Block/Plant/Sample,
  trt = ~ Treat),
  which.designs = c("first","cross"),
  data = jr.lay)

## Three-phase sensory example from Brien and Payne (1999)
## Not run:
data(Sensory3Phase.dat)
Sensory.canon <- designTwophaseAnatomies(formulae = list(
  eval= ~(Occasions/Intervals/Sittings)*Judges)/Positions,
  field= ~(Rows*(Squares/Columns))/Halfplots,
  treats= ~ Trellis*Method),
  data = Sensory3Phase.dat)

## End(Not run)
```

detect differentiate Computes the detectable difference for an experiment

Description

Computes the delta that is detectable for specified replication, power, alpha.

Usage

detect.diff(rm=5, df.num=1, df.denom=10, sigma=1, alpha=0.05, power=0.8, tol = 0.001, 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.
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 detectable difference.
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 detectable difference.

Author(s)

Chris Brien

See Also

power.exp, no.reps in package dae.

Examples

## Compute the detectable difference for a randomized complete block design ## with four treatments given power is 0.8 and alpha is 0.05.
rm <- 5
detect.diff(rm = rm, df.num = 3, df.denom = 3 * (rm - 1), sigma = sqrt(20))

---

**efficiencies**

Extracts the canonical efficiency factors from a pcanon.object or a p2canon.object.

Description

Produces a list containing the canonical efficiency factors for the joint decomposition of two or more sets of projectors (Brien and Bailey, 2009) obtained using designAnatomy or projs.2canon.
Usage

## S3 method for class 'pcanon'

```r
efficiencies(object, which = "adjusted", ...)
```

## S3 method for class 'p2canon'

```r
efficiencies(object, which = "adjusted", ...)
```

Arguments

- **object**: A `pcanon.object` or an object of class `p2canon` produced by `projs.2canon`.
- **which**: A character string, either adjusted or pairwise. For adjusted, the canonical efficiency factor are adjusted for other projectors from the same set. For pairwise, they are the unadjusted canonical efficiency factors between pairs of projectors consisting of one projector from each of two sets.
- **...**: Further arguments passed to or from other methods. Unused at present.

Value

For a `pcanon.object`, a list with a component for each component of `object`, except for the last component – for more information about the components see `pcanon.object`.

For a `p2canon` object, a list with a component for each element of the `Q1` argument from `projs.2canon`. Each component is list, each its components corresponding to an element of the `Q2` argument from `projs.2canon`.

Author(s)

Chris Brien

References


See Also

designAnatomy, summary.pcanon, proj2.efficiency, proj2.combine, proj2.eigen, pstructure in package dae, eigen.

projector for further information about this class.

Examples

```r
## PBIBD(2) from p. 379 of Cochran and Cox (1957) Experimental Designs.
## 2nd edn Wiley, New York
PBIBD2.unit <- list(Block = 6, Unit = 4)
PBIBD2.nest <- list(Unit = "Block")
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 <- designRandomize(allocated = trt,
                              recipient = PBIBD2.unit,
                              nested.recipients = PBIBD2.nest)
```
efficiency.criteria 43

# obtain combined decomposition using designAnatomy and get the efficiencies
unit.trt.canon <- designAnatomy(list(unit=~ Block/Unit, trt=~ trt), data = PBIBD2.lay)
efficiencies.pcanon(unit.trt.canon)

# obtain the projectors for each formula using pstructure
unit.struct <- pstructure(~ Block/Unit, data = PBIBD2.lay)
trt.struct <- pstructure(~ trt, data = PBIBD2.lay)

# obtain combined decomposition projs.2canon and get the efficiencies
unit.trt.p2canon <- projs.2canon(unit.struct$Q, trt.struct$Q)
efficiencies.p2canon(unit.trt.p2canon)

efficiency.criteria  Computes efficiency criteria from a set of efficiency factors

Description
Computes efficiency criteria from a set of efficiency factors.

Usage
efficiency.criteria(efficiencies)

Arguments
efficiencies  A numeric containing a set of efficiency factors.

Details
The $aefficiency$ criterion is the harmonic mean of the nonzero efficiency factors. The $mefficiency$ criterion is the mean of the nonzero efficiency factors. The $eefficiency$ criterion is the minimum of the nonzero efficiency factors. The $sefficiency$ criterion is the variance of the nonzero efficiency factors. The $xefficiency$ is the maximum of the efficiency factors. The $order$ is the order of balance and is the number of unique nonzero efficiency factors. The $dforthog$ is the number of efficiency factors that are equal to one.

Value
A list whose components are $aefficiency$, $mefficiency$, $sefficiency$, $eefficiency$, $xefficiency$, order and $dforthog$.

Author(s)
Chris Brien

See Also
proj2.efficiency, proj2.eigen, proj2.combine in package dae, eigen.
projector for further information about this class.
Examples

```r
## PBIBD(2) from p. 379 of Cochran and Cox (1957) Experimental Designs.
## 2nd edn Wiley, New York
PBIBD2.unit <- list(Block = 6, Unit = 4)
PBIBD2.nest <- list(Unit = "Block")
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 <- designRandomize(allocated = trt,
                           recipient = PBIBD2.unit,
                           nested.recipients = PBIBD2.nest)

## obtain sets of projectors
unit.struct <- pstructure(~ Block/Unit, data = PBIBD2.lay)
trt.struct <- pstructure(~ trt, data = PBIBD2.lay)

## save intrablock efficiencies
eff.inter <- proj2.efficiency(unit.struct$Q["Unit[Block]"], trt.struct$Q["trt"])

## calculate efficiency criteria
efficiency.criteria(eff.inter)
```

---

### 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**

```r
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.

**Author(s)**

Chris Brien
See Also

Extract

Examples

```r
## 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")])
```

Description

In this main-unit design, there are 24 lanes by 11 Positions, the lanes being blocked into 6 Zones of 4 lanes. The design for the main units is for assigning 75 wheat lines, of which 73 are Nested Association Mapping (NAM) wheat lines and the other two are two check lines, Scout and Gladius. A row and column design was generated with DiGGer (Coombes, 2009). For more details see the vignette daeDesignNotes.pdf.

Usage

```r
data(Exp249.munit.des)
```

Format

A data.frame containing 264 observations of 3 variables.

Source

expab

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**

```r
## 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 <- designRandomize(allocated = CRD.treat, recipient = CRD.unit, 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))

---

### Description

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

### Usage

```r
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 \( n \times n \) 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 \( n \times n \) 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 \( n \times n \) 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

```r
## 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 corresponding to B
ar1.B <- fac.ar1mat(B, 0.6)
```
fac.divide

See Also

fac.divide in package dae.

Examples

```r
## 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))
```

Description

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

Usage

```r
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.
fac.gen

Generate all combinations of several factors and, optionally, replicate them

**Description**

Generate all combinations of several factors and, optionally, replicate them.

**Usage**

```r
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 unnamed 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

```r
## 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.match

Match, for each combination of a set of columns in \( x \), the row that has the same combination in \( \text{table} \).

**Description**

Match, for each combination of a set of columns in \( x \), the rows that has the same combination in \( \text{table} \). The argument `multiples.allow` controls what happens when there are multiple matches in \( \text{table} \) of a combination in \( x \).

**Usage**

\[
fac.match(x, \text{table}, \text{col.names}, \text{nomatch} = \text{NA}\_\text{integer}_\text{-}, \text{multiples.allow} = \text{FALSE})
\]

**Arguments**

- \( x \): an R object, normally a `data.frame`, possibly a `matrix`.
- \( \text{table} \): an R object, normally a `data.frame`, possibly a `matrix`.
- \( \text{col.names} \): A character vector giving the columns in \( x \) and \( \text{table} \) that are to be matched.
- \( \text{nomatch} \): The value to be returned in the case when no match is found. Note that it is coerced to integer.
- \( \text{multiples.allow} \): A logical indicating whether multiple matches of a combination in \( x \) to those in \( \text{table} \) is allowed. If `multiples.allow` is `FALSE`, an error is generated. If `multiples.allow` is `TRUE`, the first occurrence in \( \text{table} \) is matched. This function can be viewed as a generalization to multiple vectors of the `match` function that applies to single vectors.

**Value**

A vector of length equal to \( x \) that gives the rows in \( \text{table} \) that match the combinations of \( \text{col.names} \) in \( x \). The order of the rows is the same as the order of the combinations in \( x \). The value returned if a combination is unmatched is specified in the `nomatch` argument.

**Author(s)**

Chris Brien

**See Also**

`match`
## Examples

```r
## Not run:
# A single unmatched combination
kdata <- data.frame(Expt="D197-5",
                   Row=8,
                   Column=20, stringsAsFactors=FALSE)
index <- fac.match(kdata, D197.dat, c("Expt", "Row", "Column"))

# A matched and an unmatched combination
kdata <- data.frame(Expt=c("D197-5", "D197-4"),
                   Row=c(8, 10),
                   Column=c(20, 8), stringsAsFactors=FALSE)
index <- fac.match(kdata, D197.dat, c("Expt", "Row", "Column"))

## End(Not run)
```

### Description

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

### Usage

```r
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

```r
## 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)
```

Description

Creates several factors, one for each level of nesting.fac and each of whose values are either generated within those of a level of nesting.fac or using the values of nested.fac within a levels of nesting.fac.

Usage

```r
fac.multinested(nesting.fac, nested.fac = NULL, fac.prefix = NULL,
    nested.levs = NA, nested.labs = NA,
    outlevel = 0, outlabel = "rest", ...)
```

Arguments

- **nesting.fac**  
The factor for each of whose levels a nested facor is to be generated, except one is not generated for the outlabel level.

- **nested.fac**  
The factor whose values for a level are to be used for the factor being created for that level. If nested.fac is NULL, then the values of the levels is the the list of numbers from 1 to the replication of the level of nesting.fac, represented as characters.
fac.multinested

fac.prefix The prefix to be added to a level in naming a nested factor.
nested.levs Optional vector of levels for the nested factors. Any data value that does not match a value in nested.levs will be NA in the nested factor being generated. The default value of nested.levs is the list of numbers from 1 to the replication of the levels of nesting.fac, represented as characters. When nested.fac is not NULL, nested.levs is not used.
nested.labs Optional vector of values to use as labels for the levels of the factor. The default is as.character(levels).
outlevel The level to use for all values of a new factor that does not correspond to the level of the nesting.fac to which the new factor corresponds.
outlabel The label to use the outlevel level.
... Further arguments passed to the factor call creating a new factor.

Value
A data.frame containing a factor for each level of nesting.fac.

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

Author(s)
Chris Brien

See Also
fac.gen, fac.nested in package dae, factor.

Examples

lay <- data.frame(A = factor(rep(c(1:3), c(3,6,4)), labels = letters[1:3]))
lay$B <- fac.nested(lay$A)

#Add factors for B within each level of A
lay2 <- cbind(lay, fac.multinested(lay$A))
canon2 <- designAnatomy(list(~A/(a+b+c)), data = lay2)
summary(canon2)

#Add factors for B within each level of A, but with levels and outlabel given
lay2 <- cbind(lay, fac.multinested(lay$A, nested.levs = seq(10,60,10), outlabel = "other"))
canon2 <- designAnatomy(list(~A/(a+b+c)), data = lay2)
summary(canon2)

#Replicate the combinations of A and B three times and index them with the factor sample
lay3 <- rbind(lay,lay,lay)
lay3$sample <- with(lay3, fac.nested(fac.combine(list(A,B))))
#Add factors for B within each level of A
lay4 <- cbind(lay3, fac.multinested(nesting.fac = lay$A, nested.fac = lay$B))
canon4 <- designAnatomy(list(~(A/(a+b+c))/sample), data = lay4)
summary(canon4)

#Add factors for sample within each combination of A and B
lay5 <- with(lay4, cbind(lay4,
    fac.multinested(nesting.fac = a, fac.prefix = "a"),
    fac.multinested(nesting.fac = b, fac.prefix = "b"),
    fac.multinested(nesting.fac = c, fac.prefix = "c")))
canon5 <- designAnatomy(list(~A/(a/(a1+a2+a3)+b/(b1+b2+b3+b4+b5+b6)+c/(c1+c2+c3))), data = lay5)
summary(canon5)

#Add factors for sample within each level of A
lay6 <- cbind(lay4,
    fac.multinested(nesting.fac = lay$A, nested.fac = lay$sample, fac.prefix = "samp"))
canon6 <- designAnatomy(list(~A/(a/sampa+b/sampb+c/sampc)), data = lay6)
summary(canon6)

fac.nested

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

Description

Creates a nested 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, nested.levs=NA, nested.labs=NA, ...)

Arguments

- nesting.fac: The factor within each of whose levels the created factor is to be generated.
- nested.levs: 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 nested.levs is the list of numbers from 1 to the maximum replication of the levels of nesting.fac, represented as characters.
- nested.labs: Optional vector of values to use as labels for the levels of the factor. The default is as.character(nested.levs).
- ...: Further arguments passed to the factor call creating 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. It has a different level for each of the values of the nesting.fac with the same level.

Note

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

Author(s)

Chris Brien

See Also

cfac.gen, fac.multinested in package dae, factor.

Examples

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

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

fac.recode

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

Description

Recodes the levels and values of a factor using each value in the newlevels vector to replace the corresponding value in the vector of levels of the factor.

Usage

```r
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 creating 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), labels = c("a","b"))
```

fac.sumop computes the summation matrix that produces sums corresponding to a (generalized) 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 $X \times t(X)$.

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

```r
# 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
S.AB <- fac.sumop(AB)
```

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

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

Arguments

- `factor`: The (generalized) factor for which the variance 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

```r
## 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 corresponding to B
vc.B <- fac.vcmat(B, 2)
```
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

```R
## 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 variance
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

```r
## S3 method for class 'errors'
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.

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

```r
## 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(89,88,97,94,77,92,79,81,87,87,
                      85,87,92,89,84,79,81,80,80,88)
```
## perform the analysis of variance
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

---

**get.daeTolerance**

*Gets the value of daeTolerance for the package dae*

---

**Description**

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

**Usage**

get.daeTolerance()

**Value**

The vector of two values for daeTolerance, one named `element.tol` that is used for elements of matrices and a second named `element.eigen` that is used for eigenvalues and quantities based on them, such as efficiency factors.

**Author(s)**

Chris Brien

**See Also**

`set.daeTolerance`.

**Examples**

```r
## get daeTolerance.
get.daeTolerance()
```
harmonic.mean  \hspace{1cm} \textit{Calculates the harmonic mean.}

\section*{Description}
A function to calculate the harmonic mean of a set of nonzero numbers.

\section*{Usage}
\begin{verbatim}
harmonic.mean(x)
\end{verbatim}

\section*{Arguments}
x \hspace{1cm} An object from whose elements the harmonic mean is to be computed.

\section*{Details}
All the elements of \(x\) are tested as being less than \texttt{daeTolerance}, which is initially set to \(\texttt{Machine$double.eps}^0.5\) (about 1.5E-08). The function \texttt{set.daeTolerance} can be used to change \texttt{daeTolerance}.

\section*{Value}
A numeric. Returns \texttt{Inf} if \(x\) contains a value close to zero

\section*{Examples}
\begin{verbatim}
y <- c(seq(0.1,1,0.2))
harmonic.mean(y)
\end{verbatim}

\section*{interaction.ABC.plot  \hspace{1cm} \textit{Plots an interaction plot for three factors}}

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

\section*{Usage}
\begin{verbatim}
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, ggplotFuncs = NULL, ...)
\end{verbatim}
interaction.ABC.plot

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. If the levels are numeric values stored as characters, they will be converted to numeric values for plotting. If they are actually numeric codes for nonnumeric categories and you want them plotted on a discrete scale then you should employ nonnumeric codings, such as ‘-’ and ‘+’ or ‘N’ and ‘Y’ or something similar.

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 key (legend) 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.

ggplotFuncs A list, each element of which contains the results of evaluating a ggplot function. It is created by calling the list function with a ggplot function call for each element.

... Other arguments that are passed down to ggplot methods.

Value

An object of class "ggplot", which can be plotted using print.

Author(s)

Chris Brien

See Also

fac.combine in package dae, interaction.plot.

Examples

## Not run:
## 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")

## plot for generated data
## use ?ABC.Interact.dat for data set details
data(ABC.Interact.dat)
## Add standard errors for plotting
## - here data contains a single value for each combination of A, B and C
## - need to supply name for data twice
ABC.Interact.dat$se <- rep(c(0.5,1), each=4)
interaction.ABC.plot(MOE, A, B, C, data=ABC.Interact.dat,
  ggplotFunc=list(geom_errorbar(data=ABC.Interact.dat,
    aes(ymax=MOE+se, ymin=MOE-se),
    width=0.2)))

## End(Not run)

---

### 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**

The mean of the absolute values of the elements of x is tested to determine if it is less than `daeTolerance`, which is initially set to `.Machine$double.eps ^ 0.5` (about 1.5E-08). The function `set.daeTolerance` can be used to change `daeTolerance`.

**Value**

A logical.

**Author(s)**

Chris Brien
Examples

```r
## 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 \texttt{projector}.

### Description

Tests whether an object is a valid object of class \texttt{projector}.

### Usage

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

### Arguments

- `object`: The \texttt{matrix} to be made into a projector.

### Details

The function \texttt{is.projector} tests whether the object consists of a \texttt{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 \texttt{daeTolerance}, which is initially set to \texttt{.Machine$double.eps ^ 0.5} (about 1.5E-08). The function \texttt{set.daeTolerance} can be used to change \texttt{daeTolerance}.

### Value

\texttt{TRUE} or \texttt{FALSE} depending on whether the object is a valid object of class \texttt{projector}.

### Warning

The degrees of freedom are not checked. \texttt{correct.degfree} can be used to check them.

### Author(s)

Chris Brien

### See Also

\texttt{projector}, \texttt{correct.degfree} in package \texttt{dae}.

\texttt{projector} for further information about this class.
Examples

```r
## 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)
```

Description

The systematic design for a lattice square for 49 treatments consisting of four 7 x 7 squares. For more details see the vignette `daeDesignNotes.pdf`.

Usage

```r
data(LatticeSquare_t49.des)
```

Format

A data.frame containing 196 observations of 4 variables.

Source


marginality

```
Extracts the marginality matrix (matrices) from a pstructure.object or a pcanon.object.
```

Description

Produces (i) a marginality matrix for the formula in a call to `pstructure.formula` or (ii) a list containing the marginality matrices, one for each formula in the `formulae` argument of a call to `designAnatomy`.

A marginality matrix for a set of terms is a square matrix with a row and a column for each term-aliased term. Its elements are zeroes and ones, the entry in the ith row and jth column indicates whether or not the ith term is marginal to the jth term, i.e., the column space of the ith term is a subspace of that for the jth term and so the source for the jth term will be orthogonal to that for the ith term.
Usage

## S3 method for class 'pstructure'
marginality(object, ...)
## S3 method for class 'pcanon'
marginality(object, ...)

Arguments

- **object**: A `pstructure.object` produced by `pstructure.formula` or `pcanon.object` produced by `designAnatomy`.
- **...**: Further arguments passed to or from other methods. Unused at present.

Value

- If `object` is a `pstructure.object` then a matrix containing the marginality matrix for the terms obtained from the formula in the call to `pstructure.formula`.
- If `object` is a `pcanon.object` then a list with a component for each formula, each component having a marginality matrix that corresponds to one of the formulae in the call to `designAnatomy`. The components of the list will have the same names as the components of the formulae list and so will be unnamed if the components of the latter list are unnamed.

Author(s)

Chris Brien

See Also

- `pstructure.formula`, `designAnatomy`, `summary.pcanon`, `proj2.efficiency`, `proj2.combine`, `proj2.eigen`, `pstructure` 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(Block = 6, Unit = 4)
PBIBD2.nest <- list(Unit = "Block")
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 <- designRandomize(allocated = trt, 
                            recipient = PBIBD2.unit, 
                            nested.recipients = PBIBD2.nest)

#obtain pstructure.object and extract marginality matrix
unit.struct <- pstructure(~ Block/Unit, data = PBIBD2.lay)
unit.marg <- marginality(unit.struct)

#obtain combined decomposition and extract marginality matrices
unit.trt.canon <- designAnatomy(list(unit=~ Block/Unit, trt=~ trt), data = PBIBD2.lay)
marg <- marginality(unit.trt.canon)
**Description**

Form the correlation matrix of order order whose correlations follow the ar1 pattern. The matrix is banded and has diagonal elements equal to one and the off-diagonal element in the ith row and jth column equal to \( \rho^{|i-j|} \).

**Usage**

\[
\text{mat.ar1}(\rho, \text{order})
\]

**Arguments**

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

**Value**

A banded correlation matrix whose elements follow an ar1 pattern.

**Author(s)**

Chris Brien

**See Also**

mat.I, mat.J, mat.exp, mat.gau, mat.banded, mat.ar2, mat.ar3, mat.sar2, mat.ma1, mat.ma2, mat.arma

**Examples**

\[
\text{corr} <- \text{mat.ar1}(\rho=0.4, \text{order}=4)
\]

---

**Description**

Form the correlation matrix of order order whose correlations follow the ar2 pattern. The resulting matrix is banded.

**Usage**

\[
\text{mat.ar2}(\text{ARparameters, order})
\]
Arguments

**ARparameters**  A numeric containing the two autoregressive parameter values of the process, being the weights given to the lag 1 and lag 2 response values.

**order**  The order of the matrix to be formed.

Details

The correlations in the correlation matrix, corr say, are calculated from the autoregressive parameters, ARparameters. The values in

- the diagonal (k = 1) of corr are one;
- the first subdiagonal band (k = 2) of corr are equal to ARparameters[1]/(1-ARparameters[2]);
- in subsequent diagonal bands, (k = 3:order), of corr are ARparameters[1]*corr[k-1] + ARparameters[2]*corr[k-2].

Value

A banded correlation matrix whose elements follow an ar2 pattern.

Author(s)

Chris Brien

See Also

mat.I, mat.J, mat.exp, mat.gau, mat.banded, mat.ar1, mat.ar3, mat.sar2, mat.ma1, mat.ma2, mat.arma

Examples

```r
corr <- mat.ar2(ARparameters = c(0.4, 0.2), order = 4)
```

Description

Form the correlation matrix of order order whose correlations follow the ar3 pattern. The resulting matrix is banded.

Usage

mat.ar3(ARparameters, order)
Arguments

**ARparameters**  
A numeric containing the three autoregressive parameter values of the process, being the weights given to the lag 1, lag 2 and lag 3 response values.

**order**  
The order of the matrix to be formed.

Details

The correlations in the correlation matrix, corr say, are calculated from the autoregressive parameters, ARparameters.

Let \( \omega = 1 - \text{ARparameters}[2] - \text{ARparameters}[3] \times (\text{ARparameters}[1] + \text{ARparameters}[3]) \). Then the values in

- the diagonal of corr \((k = 1)\) are one;
- the first subdiagonal band \((k = 2)\) of corr are equal to \((\text{ARparameters}[1] + \text{ARparameters}[2] \times \text{ARparameters}[3]) / \omega\);
- the second subdiagonal band \((k = 3)\) of corr are equal to \((\text{ARparameters}[1] \times (\text{ARparameters}[1] + \text{ARparameters}[3]) + \text{ARparameters}[2] \times (1 - \text{ARparameters}[2])) / \omega\);
- the subsequent subdiagonal bands, \((k = 4: \text{order})\), of corr are equal to \(\text{ARparameters}[1] \times \text{corr}[k-1] + \text{ARparameters}[2] \times \text{corr}[k-2] + \text{ARparameters}[3] \times \text{corr}[k-3]\).

Value

A banded correlation matrix whose elements follow an ar3 pattern.

Author(s)

Chris Brien

See Also

mat.I, mat.J, mat.banded, mat.exp, mat.gau, mat.ar1, mat.ar2, mat.sar2, mat.ma1, mat.ma2, mat.arma

Examples

corr <- mat.ar3(ARparameters = c(0.4, 0.2, 0.1), order = 4)
**mat arma**

*Forms an arma correlation matrix*

**Description**

Form the correlation matrix of order order whose correlations follow the arma pattern. The resulting matrix is banded.

**Usage**

```
mat arma(ARparameter, MAparameter, order)
```

**Arguments**

- **ARparameter**: A numeric value for the autoregressive parameter of the process, being the weight given to the lag 1 response values.
- **MAparameter**: A numeric value for the moving average parameter of the process, being the weight given to the lag 1 random variable.
- **order**: The order of the matrix to be formed.

**Details**

The correlations in the correlation matrix, corr say, are calculated from the correlation parameters, ARparameters. The values in

- the diagonal (k = 1) of corr are one;
- the first subdiagonal band (k = 2) of corr are equal to ARparameters[1]/(1-ARparameters[2]);
- in subsequent diagonal bands, (k = 3:order), of corr are ARparameters[1]*corr[k-1] + ARparameters[2]*corr[k-2].

**Value**

A banded correlation matrix whose elements follow an arma pattern.

**Author(s)**

Chris Brien

**See Also**

`mat.I, mat.J, mat.exp, mat.gau, mat.banded, mat.ar1, mat.ar3, mat.sar2, mat.ma1, mat.ma2`

**Examples**

```
corr <- mat arma(ARparameter = 0.4, MAparameter = -0.2, order = 4)
```
Form a banded matrix from a vector of values

Description

Takes the first value in x and places it down the diagonal of the matrix. Takes the second value in x and places it down the first subdiagonal, both below and above the diagonal of the matrix. The third value is placed in the second subdiagonal and so on, until the bands for which there are elements in x have been filled. All other elements in the matrix will be zero.

Usage

mat.banded(x, nrow, ncol)

Arguments

- **x**: A numeric containing the values for each band from 1 to the length of x.
- **nrow**: The number of rows in the banded matrix being formed.
- **ncol**: The number of columns in the banded matrix being formed.

Value

An nrow x ncol matrix.

Author(s)

Chris Brien

See Also

matmult, mat.ar1, mat.ar2, mat.ar3, mat.sar2, mat.exp, mat.gau, mat.ml, mat.ma2, mat.arma, mat.I, mat.J

Examples

```r
m <- mat.banded(c(1,0.6,0.5), 5,5)
m <- mat.banded(c(1,0.6,0.5), 3,4)
m <- mat.banded(c(1,0.6,0.5), 4,3)
```
**Description**

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

The method employed uses the \texttt{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**

\texttt{mat.dirprod(A, B)}

**Arguments**

- \( A \): The left-hand \texttt{matrix} in the product.
- \( B \): The right-hand \texttt{matrix} in the product.

**Value**

An \( mp \times nq \) \texttt{matrix}.

**Author(s)**

Chris Brien

**See Also**

\texttt{matmult, mat.dirprod}

**Examples**

\begin{verbatim}
\end{verbatim}
mat.dirsum

Forms the direct sum of a list of matrices

Description

The direct sum is the partitioned matrices whose diagonal submatrices are the matrices from which
the direct sum is to be formed and whose off-diagonal submatrices are conformable matrices of ze-
roes. The resulting matrix is \( m \times n \), where \( m \) is the sum of the numbers of rows of the contributing
matrices and \( n \) is the sum of their numbers of columns.

Usage

mat.dirsum(matrices)

Arguments

matrices  A list, each of whose component is a matrix.

Value

An \( m \times n \) matrix.

Author(s)

Chris Brien

See Also

mat.dirprod, matmult

Examples

m1 <- matrix(1:4, nrow=2)
m2 <- matrix(11:16, nrow=3)
m3 <- diag(1, nrow=2, ncol=2)
dsum <- mat.dirsum(list(m1, m2, m3))
\textbf{mat.exp} \hspace{5cm} \textit{Forms an exponential correlation matrix}

\textbf{Description}
Form the correlation matrix of order equal to the length of coordinates. The matrix has diagonal elements equal to one and the off-diagonal element in the $i$th row and $j$th column equal to $\rho^k$ where $k = |coordinate[i] - coordinate[j]|$.

\textbf{Usage}
\begin{verbatim}
mat.exp(rho, coordinates)
\end{verbatim}

\textbf{Arguments}
\begin{itemize}
\item \texttt{rho} \hspace{1cm} The correlation for points a distance of one apart.
\item \texttt{coordinates} \hspace{1cm} The coordinates of points whose correlation matrix is to be formed.
\end{itemize}

\textbf{Value}
A correlation matrix whose elements depend on the power of the absolute distance apart.

\textbf{Author(s)}
Chris Brien

\textbf{See Also}
\begin{verbatim}
mat.I, mat.J, mat.banded, mat.ar1, mat.ar2, mat.ar3, mat.sar2, mat.ma1, mat.ma2, mat.arma, mat.gau
\end{verbatim}

\textbf{Examples}
\begin{verbatim}
corr <- mat.exp(coordinates=c(3:6, 9:12, 15:18), rho=0.1)
\end{verbatim}

\textbf{mat.gau} \hspace{5cm} \textit{Forms an exponential correlation matrix}

\textbf{Description}
Form the correlation matrix of order equal to the length of coordinates. The matrix has diagonal elements equal to one and the off-diagonal element in the $i$th row and $j$th column equal to $\rho^k$ where $k = (coordinate[i] - coordinate[j])^2$.

\textbf{Usage}
\begin{verbatim}
mat.gau(rho, coordinates)
\end{verbatim}
Arguments

rho  The correlation for points a distance of one apart.
coordinates The coordinates of points whose correlation *matrix* is to be formed.

Value

A correlation *matrix* whose elements depend on the power of the absolute distance apart.

Author(s)

Chris Brien

See Also

mat.I, mat.J, mat.banded, mat.ar1, mat.ar2, mat.ar3, mat.sar2, mat.mal, mat.ma2, mat.arma, mat.exp

Examples

corr <- mat.gau(coordinates=c(3:6, 9:12, 15:18), rho=0.1)

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.

Author(s)

Chris Brien

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.

Author(s)

Chris Brien

See Also

mat.I, mat.ar1

Examples

col.J <- mat.J(order=4)
**mat.ma1**

Forms an ma1 correlation matrix

Description
Form the correlation matrix of order order whose correlations follow the ma1 pattern. The matrix is banded and has diagonal elements equal to one and subdiagonal element equal to 

\[-MAparameter / (1 + MAparameter^2)\]

Usage

mat.ma1(MAparameter, order)

Arguments

- **MAparameter**
  The moving average parameter, being the weight applied to the lag 1 random perturbation.
- **order**
  The order of the matrix to be formed.

Value

A banded correlation matrix whose elements follow an ma1 pattern.

Author(s)

Chris Brien

See Also

mat.I, mat.J, mat.exp, mat.gau, mat.banded, mat.ar2, mat.ar3, mat.sar2, mat.ma2, mat arma

Examples

```
corr <- mat.ma1(MAparameter=0.4, order=4)
```

---

**mat.ma2**

Forms an ma2 correlation matrix

Description
Form the correlation matrix of order order whose correlations follow the ma2 pattern. The resulting matrix is banded.

Usage

mat.ma2(MAparameters, order)
Arguments

MAparameters A numeric containing the two moving average parameter values of the process, being the weights given to the lag 1 and lag 2 random perturbations.

order The order of the matrix to be formed.

Details

The correlations in the correlation matrix, corr say, are calculated from the moving average parameters, MAparameters. The values in

- the diagonal (k = 1) of corr are one;
- the first subdiagonal band (k = 2) of corr are equal to -MAparameters[1]*(1-MAparameters[2]) / div;
- the second subdiagonal band (k = 3) of corr are equal to -MAparameters[2] / div;
- in subsequent diagonal bands, (k = 4:order), of corr are zero,


Value

A banded correlation matrix whose elements follow an ma2 pattern.

Author(s)

Chris Brien

See Also

mat.I, mat.J, mat.exp, mat.gau, mat.banded, mat.ar1, mat.ar3, mat.sar2, mat.ma1, mat.arma

Examples

corr <- mat.ma2(MAparameters = c(0.4, -0.2), order = 4)

mat.ncssvar

Calculates the variance matrix of the random effects for a natural cubic smoothing spline

Description

Calculates the variance matrix of the random effects for a natural cubic smoothing spline. It is the tri-diagonal matrix $G_s$ given by Verbyla et al., (1999) multiplied by the variance component for the random spline effects.

Usage

mat.ncssvar(sigma2s = 1, knot.points, print = FALSE)
Arguments

**sigma2s**  
A numeric giving the value of the variance component for the random spline effects. The smoothing parameter is then the inverse of the ratio of this component to the residual variance.

**knot.points**  
A numeric giving the values of the knots point used in fitting the spline. These must be ordered in increasing order.

**print**  
A logical indicating whether to print the matrix.

Value

A matrix containing the variances and covariances of the random spline effects.

Author(s)

Chris Brien

References


See Also

Zncsspline.

Examples

Gs <- mat.ncssvar(knot.points = 1:10)

---

**mat.sar**  
Forms an sar correlation matrix

Description

Form the correlation matrix of order order whose correlations follow the sar pattern. The resulting matrix is banded.

Usage

mat.sar(SARparameter, order)

Arguments

**SARparameter**  
A numeric containing the single value of the parameter from which the correlations are calculated.

**order**  
The order of the matrix to be formed.
Details

The values of the correlations in the correlation matrix, \( \text{corr} \) say, are calculated from the SARparameter, gamma as follows. The values in

- the diagonal of \( \text{corr} \) \((k = 1)\) are one;
- the first subdiagonal band \((k = 2)\) of \( \text{corr} \) are equal to \( \text{gamma}/(1 + (\text{gamma} \times \text{gamma} / 4)) \);
- the subsequent subdiagonal bands, \((k = 3: \text{order})\), of \( \text{corr} \) are equal to \( \text{gamma} \times \text{corr}[k-1] - (\text{gamma} \times \text{gamma}/4) \times \text{corr}[k-2] \).

Value

A banded correlation matrix whose elements follow an sar pattern.

Author(s)

Chris Brien

See Also

`mat.I, mat.J, mat.banded, mat.exp, mat.gau, mat.ar1, mat.ar2, mat.ar3, mat.sar2, mat.ma1, mat.ma2, mat arma`

Examples

```r
corr <- mat.sar(SARparameter = -0.4, order = 4)
```

---

**Description**

Form the correlation matrix of order \( \text{order} \) whose correlations follow the sar2 pattern, a pattern used in crop competition models. The resulting matrix is banded and is a constrained AR3 matrix.

Usage

```r
mat.sar2(gamma, order, print = NULL)
```

Arguments

- `gamma`: A numeric containing the two values of \( \text{gamma} \), being parameters linked with spatial dependence and competition.
- `order`: The order of the matrix to be formed.
- `print`: A character giving the object to be printed. Currently, only the calculated values of the ar3 parameters can be printed. If NULL, nothing is printed.
Details

The values of the AR3 parameters, phi, are calculated from the gammas as follows:

\[
\begin{align*}
\phi[1] &= \gamma[1] + 2 \times \gamma[2]; \\
\phi[2] &= -\gamma[2] \times (2\gamma[2] + \gamma[1]); \\
\phi[3] &= \gamma[1] \times \gamma[2] \times \gamma[2].
\end{align*}
\]

Then the correlations in the correlation matrix, \( \text{corr} \) say, are calculated from the correlation parameters, phi. Let \( \omega = 1 - \phi[2] - \phi[3] \times (\phi[1] + \phi[3]) \). Then the values in

- the diagonal of \( \text{corr} \) (\( k = 1 \)) are one;
- the first subdiagonal band (\( k = 2 \)) of \( \text{corr} \) are equal to \( (\phi[1] + \phi[2] \times \phi[3]) / \omega \);
- the second subdiagonal band (\( k = 3 \)) of \( \text{corr} \) are equal to \( (\phi[1] \times (\phi[1] + \phi[3]) + \phi[2] \times (1 - \phi[2])) / \omega \);
- the subsequent subdiagonal bands, (\( k = 4: \text{order} \)), of \( \text{corr} \) are equal to \( \phi[1] \times \text{corr}[k-1] + \phi[2] \times \text{corr}[k-2] + \phi[3] \times \text{corr}[k-3] \).

Value

A banded correlation matrix whose elements follow an sar2 pattern.

Author(s)

Chris Brien

See Also

\mat.I, \mat.J, \mat.banded, \mat.exp, \mat.gau, \mat.ar1, \mat.ar2, \mat.ar3, \mat.sar, \mat.ma1, \mat.ma2, \mat.arma

Examples

\[
\begin{align*}
corr &\leftarrow \text{mat.sar2}(\gamma = c(-0.4, 0.2), \text{order} = 4) \\
corr &\leftarrow \text{mat.sar2}(\gamma = c(-0.4, 0.2), \text{order} = 4, \text{print} = \text{"ar3"})
\end{align*}
\]

\mat.Vpred \quad Calculates the variances of a set of predicted effects from a mixed model

Description

For \( n \) observations, \( w \) effects to be predicted, \( f \) nuisance fixed effects and \( r \) nuisance random effects, the variances of a set of predicted effects is calculated using the incidence matrix for the effects to be predicted and, optionally, a variance matrix of the effects, an incidence matrix for the nuisance fixed factors and covariates, the variance matrix of the nuisance random effects in the mixed model and the residual variance matrix.

This function has been superseded by \mat.Vpredicts, which allows the use of both matrices and formulae.
Usage

mat.Vpred(W, Gg = 0, X = matrix(1, nrow = nrow(W), ncol = 1), Vu = 0, R, eliminate)

Arguments

W
The n x w incidence matrix for the w effects to be predicted.

Gg
The w x w variance matrix of the w effects to be predicted. If the effects to be predicted are fixed, set to 0.

X
The n x f incidence matrix for the f nuisance fixed factors and covariates. The default is a column vector of ones.

Vu
The n x r variance matrix of the r nuisance random effects. If there are none, set to zero.

R
The residual variance matrix.

eliminate
The n x n projector onto the subspace corresponding to the effects to be eliminated from the information matrix prior to inverting it to form the variance matrix of the predicted effects. It is only appropriate to use this option when the effects to be predicted are fixed.

Details

Firstly the information matrix is calculated as

\[ A <- t(W)^t \cdot Vinv \cdot W + ginv(Gg) - A^t \cdot ginv(t(X) \cdot Vinv \cdot X) \cdot A, \]

where \( Vinv \) is the unique Moore-Penrose inverse of \( Vu + R \), \( A = t(W) \cdot Vinv \cdot X \) and \( ginv(B) \) is the unique Moore-Penrose inverse of \( B \) formed using the eigendecomposition of \( B \).

If eliminate is set and the effects to be predicted are fixed then the reduced information matrix is calculated as \( A <- (I - \text{eliminate}) \cdot Vinv \cdot (I - \text{eliminate}) \).

Finally, the variance of the predicted effects is calculated: \( Vpred <- ginv(A) \).

Value

A w x w matrix containing the variances and covariances of the predicted effects.

Author(s)

Chris Brien

References


See Also

designAmeasures, mat.Vpredicts.
Examples

## Reduced example from Smith et al. (2015)
## Generate two-phase design
mill.fac <- fac.gen(list(Mrep = 2, Mday = 2, Mord = 3))
field.lay <- fac.gen(list(Frep = 2, Fplot = 4))
field.lay$Variety <- factor(c("D", "E", "Y", "W", "G", "D", "E", "M"),
                        levels = c("Y", "W", "G", "M", "D", "E"))
start.design <- cbind(mill.fac, field.lay[c(3,4,5,8,1,7,3,4,5,8,6,2),])
rownames(start.design) <- NULL

## Set up matrices
n <- nrow(start.design)
W <- model.matrix(~ -1+ Variety, start.design)
ng <- ncol(W)
Gg<- diag(1, ng)
Vu <- with(start.design, fac.vcmat(Mrep, 0.3) +
          fac.vcmat(fac.combine(list(Mrep, Mday)), 0.2) +
          fac.vcmat(Frep, 0.1) +
          fac.vcmat(fac.combine(list(Frep, Fplot)), 0.2))
R <- diag(1, n)

## Calculate the variance matrix of the predicted random Variety effects
Vp <- mat.Vpred(W = W, Gg = Gg, Vu = Vu, R = R)
designAmeasures(Vp)

## Calculate the variance matrix of the predicted fixed Variety effects,
## eliminating the grand mean
Vp.reduc <- mat.Vpred(W = W, Gg = 0, Vu = Vu, R = R,
                       eliminate = projector(matrix(1, nrow = n, ncol = n)/n))
designAmeasures(Vp.reduc)

mat.Vpredicts

Calculates the variances of a set of predicted effects from a mixed
model, based on supplied matrices or formulae.

Description

For n observations, w effects to be predicted, f nuisance fixed effects, r nuisance random effects
and n residuals, the variances of a set of predicted effects is calculated using the incidence matrix for
the effects to be predicted and, optionally, a variance matrix of these effects, an incidence matrix for
the nuisance fixed factors and covariates, the variance matrix of the nuisance random effects and the
residual variance matrix. The matrices can be supplied directly or using formulae and a matrix
specifying the variances of the nuisance random effects. The difference between mat.Vpredicts
and mat.Vpred is that the former has different names for equivalent arguments and the latter does
not allow for the use of formulae.

Usage

mat.Vpredicts(target, Gt = 0, fixed = ~ 1, random, G, R, design,
              eliminate, keep.order = TRUE, result = "variance.matrix")
Arguments

target The \( n \times w \) incidence matrix for the \( w \) effects targeted for prediction, or a formula from which the matrix can be generated.

\( G_t \) The value of the variance component for the targeted effects or the \( w \times w \) variance matrix of the \( w \) targeted effects. If the targeted effects are fixed, set \( G_t \) to 0.

fixed The \( n \times f \) incidence matrix for the \( f \) nuisance fixed effects and covariates, or a formula from which the matrix can be generated. The default is a formula for an intercept-only model.

random The \( n \times r \) incidence matrix for the \( r \) nuisance random effects, or a formula from which the matrix can be generated. The default is 0, which implies that there are no random effects.

\( G \) The \( r \times r \) variance matrix of the \( r \) nuisance random effects. This term only needs to be set if random is a formula. Then it is set to a list, in which each component is either a single value or a matrix; there needs to be a component for each term in the expanded formula, with the order of the terms and components matching. If it is a single value, a diagonal matrix of dimension equal to the product of the numbers of levels of the factors in its term. If it is a matrix, its dimension must be equal to the product of the numbers of levels of the factors in its term.

\( R \) The \( n \times n \) residual variance matrix. If \( R \) is not set in the call, then it is set to the identity matrix.

design A data.frame containing the design to be used in an experiment from which predictions are to be obtained. It is not required when the only formula specified is an intercept-only model.

eliminate The \( n \times n \) projector onto the subspace corresponding to the effects to be eliminated from the information matrix prior to inverting it to form the variance matrix of the predicted effects. It is only appropriate to use this option when the effects to be predicted are fixed.

keep.order A logical indicating whether the terms should keep their position in the expanded formula projector, or reordered so that main effects precede two-factor interactions, which precede three-factor interactions and so on.

result A character indicating which matrix is to be returned: variance.matrix or information.matrix.

Details

The mixed model for which the predictions are to be obtained is of the form

\[
Y = X\beta + Ww + Zu + e,
\]

where \( W \) is the incidence matrix for the target predicted effects \( w \), \( X \) is the incidence matrix for the fixed nuisance effects \( \beta \), \( Z \) is the incidence matrix for the random nuisance effects \( u \), \( e \) are the residuals; the \( u \) are assumed to have variance matrix \( G \) so that their contribution to the variance matrix for \( Y \) is \( Vu = ZGZ^T \) and \( e \) is assumed to have variance matrix \( R \). If the target effects are random then the variance matrix for \( w \) is \( G_t \) so that their contribution to the variance matrix for \( Y \) is \( W G_t W^T \).

As described in Hooks et al. (2009, Equation 19), the information matrix is calculated as

\[
A <- t(W) %*% Vinv %*% W + ginv(Gg) - A %*% ginv(t(X) %*% Vinv %*% X) %*% t(A),
\]

where \( Vinv <- ginv(Vu \)
mat.Vpredicts

+ R), \( A = \text{t}(W) \times \text{ginv}(V) \times X \) and \( \text{ginv}(B) \) is the unique Moore-Penrose inverse of \( B \) formed using the eigendecomposition of \( B \).

Then, if \( \text{eliminate} \) is set and the effects to be predicted are fixed then the reduced information matrix is calculated as \( A = (I - \text{eliminate}) \times \text{ginv}(I - \text{eliminate}) \).

Finally, if \( \text{result} \) is set to \( \text{variance.matrix} \), the variance of the predicted effects is calculated: \( \text{Vpred} = \text{ginv}(A) \) and returned; otherwise the information matrix \( A \) is returned. The rank of the matrix to be returned is obtained via a singular value decomposition of the information matrix, it being the number of nonzero eigenvalues. An eigenvalue is regarded as zero if it is less than \( \text{daeTolerance} \), which is initially set to \( \text{Machine}^\text{double.eps} \times 0.5 \) (about 1.5E-08). The function \( \text{set.daeTolerance} \) can be used to change \( \text{daeTolerance} \).

Value

A \(\times\times\) matrix containing the variances and covariances of the predicted effects or the information matrix for the effects, depending on the setting of \( \text{result} \). The matrix has its rank as an attribute.

Author(s)

Chris Brien

References


See Also

designAmeasures, mat.Vpred.

Examples

```r
## Reduced example from Smith et al. (2015)
## Generate two-phase design
mill.fac <- fac.gen(list(Mrep = 2, Mday = 2, Mord = 3))
field.lay <- fac.gen(list(Frep = 2, Fplot = 4))
field.lay$Variety <- factor(c("D","E","Y","W","G","D","E","M"),
levels = c("Y","W","G","M","D","E"))
start.design <- cbind(mill.fac, field.lay[,c(3,4,5,8,1,7,3,4,5,8,6,2),])
rownames(start.design) <- NULL

## Set gammas
gammas <- c(1, 0.1, 0.2, 0.3, 0.2, 1)
names(gammas) <- terms

## Specify matrices to calculate the variance matrix of the predicted fixed Variety effects
W <- model.matrix(~ -1 + Variety, start.design)
```
McIntyre TMV.dat

```r
Vu <- with(start.design, fac.vcmat(Mrep, gammas["Mrep"])) +
    fac.vcmat(fac.combine(list(Mrep,Mday)), gammas["Mrep:Mday"]) +
    fac.vcmat(Frep, gammas["Frep"]) +
    fac.vcmat(fac.combine(list(Frep,Fplot)), gammas["Frep:Fplot"]))
R <- diag(1, nrow(start.design))

## Calculate variance matrix
Vp <- mat.Vpredicts(target = W, random=Vu, R=R)

## Calculate the variance matrix of the predicted random Variety effects using formulae
Vp <- mat.Vpredicts(target = ~ -1 + Variety, Gt = 1,
    fixed = ~ 1,
    random = ~ -1 + Mrep/Mday + Frep/Fplot,
    G = as.list(gammas[c(4,5,2,3)]),
    R = R, design = start.design)
designAmeasures(Vp)

## Calculate the variance matrix of the predicted fixed Variety effects,
## eliminating the grand mean
n <- nrow(start.design)
Vp.reduc <- mat.Vpredicts(target = ~ -1 + Variety,
    random = ~ -1 + Mrep/Mday + Frep/Fplot,
    G = as.list(gammas[c(4,5,2,3)]),
    eliminate = projector(matrix(1, nrow = n, ncol = n)/n),
    design = start.design)
designAmeasures(Vp.reduc)
```

---

**Description**

McIntyre (1955) reports an investigation of the effect of four light intensities on the synthesis of tobacco mosaic virus in leaves of tobacco *Nicotiana tabacum* var. Hickory Pryor. It is a two-phase experiment: the first phase is a treatment phase, in which the four light treatments are randomized to the tobacco leaves, and the second phase is an assay phase, in which the tobacco leaves are randomized to the half-leaves of assay plants. For more details see the vignette `daeDesignNotes.pdf`.

**Usage**

```r
data(McIntyreTMV.dat)
```

**Format**

A data.frame containing 196 observations of 4 variables.

**Source**

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

```r
## 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 levels 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=0.1, 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 list containing nreps, a single numeric value containing the computed number of pure replicates, and power, a single numeric value containing the power for the computed number of pure replicates.

Author(s)

Chris Brien
See Also

`power.exp, detect.diff` in package `dae`.

Examples

```r
## Compute the number of replicates (blocks) required for a randomized
c## 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)
```

---

Oats.dat

*Data for an experiment to investigate nitrogen response of 3 oats varieties*

Description

Yates (1937) describes a split-plot experiment that investigates the effects of three varieties of oats and four levels of Nitrogen fertilizer. The varieties are assigned to the main plots using a randomized complete block design with 6 blocks and the nitrogen levels are randomly assigned to the subplots in each main plot.

The columns in the data frame are: Blocks, Wplots, Subplots, Variety, Nitrogen, xNitrogen, Yield. The column xNitrogen is a numeric version of the factor Nitrogen. The response variable is Yield.

Usage

```r
data(Oats.dat)
```

Format

A data.frame containing 72 observations of 7 variables.

Author(s)

Chris Brien

Source

Description

An object of class `p2canon` that contains information derived from two formulae using `projs.2canon`.

Value

A list of class `p2canon`. It has two components: `decomp` and `aliasing`. The `decomp` component is composed as follows:

- It has a component for each component of `Q1`.
- Each of the components for `Q1` is a list; each of these lists has one component for each of `Q2` and a component `Pres`.
- Each of the `Q2` components is a list of three components: `pairwise`, `adjusted` and `Qproj`. These components are based on an eigenanalysis of the relationship between the projectors for the parent `Q1` and `Q2` components.
  1. Each `pairwise` component is based on the nonzero canonical efficiency factors for the joint decomposition of the two parent projectors (see `proj2.eigen`).
  2. An `adjusted` component is based on the nonzero canonical efficiency factors for the joint decomposition of the `Q1` component and the `Q2` component, the latter adjusted for all `Q2` projectors that have occurred previously in the list.
  3. The `Qproj` component is the adjusted projector for the parent `Q2` component.
- The `pairwise` and `adjusted` components have the following components: `efficiencies`, `aefficiency`, `mefficiency`, `sefficiency`, `eefficiency`, `xefficiency`, `order` and `dforthog` – for details see `efficiency.criteria`.

The `aliasing` component is a data.frame describing the aliasing between terms corresponding to two `Q2` projectors when estimated in subspaces corresponding to a `Q1` projector.

Author(s)

Chris Brien

See Also

`projs.2canon`, `designAnatomy`, `pcanon.object`. 
Description

An object of class \texttt{pcanon} that contains information derived from several formulae using \texttt{designAnatomy}.

Value

A list of class \texttt{pcanon} that has four components: (i) $Q$, (ii) terms, (iii) sources, (iv) marginality, and (v) aliasing. Each component is a list with as many components as there are formulae in the \texttt{formulae} list supplied to \texttt{designAnatomy}.

The $Q$ list is made up of the following components:

1. The first component is the joint decomposition of two structures derived from the first two formulae, being the \texttt{p2canon.object} produced by \texttt{projs.2canon}.
2. Then there is a component for each further formulae; it contains the \texttt{p2canon.object} obtained by applying \texttt{projs.2canon} to the structure for a formula and the already established joint decomposition of the structures for the previous formulae in the \texttt{formulae}.
3. The last component contains the list of the projectors that give the combined canonical decomposition derived from all of the formulae.

The terms, sources, marginality and aliasing lists have a component for each formula in the \texttt{formulae} argument to \texttt{designAnatomy}. Each component of the terms and sources lists has a character vector containing the terms or sources derived from its formula. For the marginality component, each component is the marginality matrix for the terms derived from its formula. For the aliasing component, each component is the aliasing data.frame for the source derived from its formula. The components of these four lists are produced by \texttt{pstructure.formula} and are copied from the \texttt{pstructure.object} for the formula. The names of the components of these four lists will be the names of the components in the \texttt{formulae} list.

The object has the attribute \texttt{labels}, which is set to "terms" or "sources" according to which of these were used to label the projectors when the object was created.

Author(s)

Chris Brien

See Also

\texttt{designAnatomy}, \texttt{p2canon.object}. 
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, detect.diff in package dae.

Examples

```r
## 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.aliasing  

Print an aliasing data.frame

Description

Prints an aliasing data.frame.

Usage

## S3 method for class 'aliasing'
print(x, which.criteria = c("aeficiency","eeficiency","order"), ...)  

Arguments

x The data.frame that is also of class aliasing and is to be printed.
which.criteria A character vector nominating the efficiency criteria to be included in the summary of aliasing between terms. It can be none, all or some combination of aeficiency, mefficiency, sefficiency, eefficiency, xefficiency, order and dforthog – for details see efficiency.criteria. If none, no criteria are printed.
... Further arguments passed to the print method for data.frame.

Author(s)

Chris Brien

See Also

print, print.default, show.

Examples

## Generate a data.frame with 3 factors length 12
pseudo.lay <- data.frame(pl = factor(1:12),
ab = factor(rep(1:4, times=3)),
a = factor(rep(1:2, times=6)))

## create a pstructure object
trt.struct <- pstructure(~ ab+a, data = pseudo.lay)

## print the object either using the Method function, the generic function or show
print.aliasing(trt.struct$aliasing)
print(trt.struct$aliasing, which.criteria = "none")
trt.struct$aliasing
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.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
print.pstructure

Prints a pstructure.object

Description

Prints a pstructure.object, which is of class pstructure. The df, terms and sources are coerced into a data.frame and printed; the marginality matrix is printed separately.

Usage

## S3 method for class 'pstructure'
print(x, which.criteria = c("aefficiency", "eefficiency", "order"), ...)

Arguments

x

The pstructure.object, which is of class pstructure and is to be printed.

which.criteria

A character vector nominating the efficiency criteria to be included in the summary of aliasing between terms. It can be none, all or some combination of aefficiency, mefficiency, sefficiency, eefficiency, xefficiency, order and dforthog – for details see efficiency.criteria. If none, no criteria are printed.

...

Further arguments passed to or from other methods.

Author(s)

Chris Brien

See Also

print, print.default, show.

Examples

## Generate a data.frame with 4 factors, each with three levels, in standard order
ABCD.lay <- fac.gen(list(A = 3, B = 3, C = 3, D = 3))

## create a pstructure object based on the formula ((A*B)/C)*D
ABCD.struct <- pstructure.formula(~ ((A*B)/C)*D, data = ABCD.lay)

## print the object either using the Method function, the generic function or show
print.pstructure(ABCD.struct)
print(ABCD.struct)
ABCD.struct
Description

Prints a `summary.p2canon` object, which is also a `data.frame`, in a pretty format.

Usage

```r
## S3 method for class 'summary.p2canon'
print(x, ...)  
```

Arguments

- `x`: A `summary.p2canon` object.
- `...`: further arguments passed to `print`.

Value

No value is returned.

Author(s)

Chris Brien

See Also

`summary.p2canon`

Examples

```r
## PBIBD(2) from p. 379 of Cochran and Cox (1957) Experimental Designs.  
## 2nd edn Wiley, New York
PBIBD2.unit <- list(Block = 6, Unit = 4)  
PBIBD2.nest <- list(Unit = "Block")  
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 <- designRandomize(allocated = trt,  
  recipient = PBIBD2.unit,  
  nested.recipients = PBIBD2.nest)

##obtain projectors using pstructure
unit.struct <- pstructure(~ Block/Unit, data = PBIBD2.lay)  
trt.struct <- pstructure(~ trt, data = PBIBD2.lay)

##obtain combined decomposition and print summary
unit.trt.p2canon <- projs.2canon(unit.struct$Q, trt.struct$Q)  
summ <- summary(unit.trt.p2canon)  
print(summ)
```
print.summary.pcanon  

Prints the values in an summary.pcanon object

Description
Prints a summary.pcanon object, which is also a data.frame, in a pretty format.

Usage

## S3 method for class 'summary.pcanon'
print(x, aliasing.print = TRUE, ...)

Arguments

x  
A summary.pcanon object.

aliasing.print  
A logical indicating whether the aliasing between sources is to be printed.  
Ignored for legacy summary.pcanon objects resulting from versions prior to 3.0-0  
and so using projs.cannon

...  
further arguments passed to print.

Value
No value is returned.

Author(s)
Chris Brien

See Also
summary.pcanon

Examples

## PBIBD(2) from p. 379 of Cochran and Cox (1957) Experimental Designs.  
## 2nd edn Wiley, New York
PBIBD2.unit <- list(Block = 6, Unit = 4)  
PBIBD2.nest <- list(Unit = "Block")  
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 <- designRandomize(allocated = trt,  
recipient = PBIBD2.unit,  
nested.recipients = PBIBD2.nest)

## obtain combined decomposition and summarize
unit.trt.canon <- designAnatomy(list(unit=~ Block/Unit, trt=~ trt),  
data = PBIBD2.lay)
summ <- summary(unit.trt.canon, which = c("aeff","eeff","order"))
print(summ)
proj2.combine

Compute the projection and Residual operators for two, possibly nonorthogonal, projectors

Description

The canonical relationship between a pair of projectors is established by decomposing 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.combine(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 $Q1 \times Q2 \times Q1$ (James and Wilkinson, 1971). An eigenvalue is regarded as zero if it is less than daeTolerance, which is initially set to $\text{.Machine}$double$\text{.eps}^0.5$ (about 1.5E-08). 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), if the number of distinct canonical efficiency factors is less than 10. If this fails to produce a projector or the number of distinct canonical efficiency factors is 10 or more, equation 5.3 of Payne and Tobias (1992) is used to obtain Qres. In this latter case, $Q\text{res} = Q1 - Q1 \times \text{ginv}(Q2 \times Q1 \times Q2) \times Q1$. 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.
computes the canonical efficiency factors for the joint decomposition of two projectors (James and Wilkinson, 1971).
**proj2.efficiency**

**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 .Machine$double.eps ^ 0.5 (about 1.5E-08). The function set.daeTolerance can be used to change daeTolerance.

**Value**

A vector containing the nonzero canonical efficiency factors.

**Author(s)**

Chris Brien

**References**


**See Also**

efficiency.criteria, proj2.eigen, proj2.combine in package dae.eigen.

projector for further information about this class.

**Examples**

```r
## PBIBD(2) from p. 379 of Cochran and Cox (1957) Experimental Designs.
## 2nd edn Wiley, New York
PBIBD2.unit <- list(Block = 6, Unit = 4)
PBIBD2.nest <- list(Unit = "Block")
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 <- designRandomize(allocated = trt, 
recipient = PBIBD2.unit, 
nested.recipients = PBIBD2.nest)

## obtain sets of projectors
unit.struct <- pstructure(~ Block/Unit, data = PBIBD2.lay)
trt.struct <- pstructure(~ trt, data = PBIBD2.lay)

## save intrablock efficiencies
eff.intra <- proj2.efficiency(unit.struct$Q["Block"], trt.struct$Q["trt"])
```
Description

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

Usage

proj2.eigen(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 .Machine$double.eps ^ 0.5 (about 1.5E-08). 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


See Also

proj2.efficiency, proj2.combine in package dae, eigen.
projector for further information about this class.
Examples

```r
## PBIBD(2) from p. 379 of Cochran and Cox (1957) Experimental Designs.
## 2nd edn Wiley, New York
PBIBD2.unit <- list(Block = 6, Unit = 4)
PBIBD2.nest <- list(Unit = "Block")
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 <- designRandomize(allocated = trt,
                              recipient = PBIBD2.unit,
                              nested.recipients = PBIBD2.nest)

## obtain sets of projectors
unit.struct <- pstructure(~ Block/Unit, data = PBIBD2.lay)
trt.struct <- pstructure(~ trt, data = PBIBD2.lay)

## obtain intra- and inter-block decompositions
decomp.inter <- proj2.eigen(unit.struct$Q[["Block"]], trt.struct$Q[["trt"]])
decomp.intra <- proj2.eigen(unit.struct$Q[["Unit[block]"], trt.struct$Q[["trt"]])

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

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 `.Machine$double.eps ^ 0.5` (about 1.5E-08). The function `set.daeTolerance` can be used to change `daeTolerance`.
Value
An object of Class "projector" that consists of a square, symmetric, 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)

---

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


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
A canonical analysis of the relationships between two sets of projectors

Description

Computes the canonical efficiency factors for the joint decomposition of two structures or sets of mutually orthogonally projectors (Brien and Bailey, 2009), orthogonalizing projectors in the Q2 list to those earlier in the list of projectors with which they are partially aliased. The results can be summarized in the form of a skeleton ANOVA table.

Usage

projs.2canon(Q1, Q2)

Arguments

Q1
- A list whose components are objects of class "projector".

Q2
- A list whose components are objects of class "projector".

Details

Two loops, one nested within the other, are performed. The first cycles over the components of Q1 and the nested loop cycles over the components of Q2. The joint decomposition of the two projectors in each cycle, one from Q1 (say Q1[[i]]) and the other from Q2 (say Q2[[j]]) is obtained using proj2combine. In particular, the nonzero canonical efficiency factors for the joint decomposition of the two projectors is obtained. The nonzero canonical efficiency factors are the nonzero eigenvalues of Q1[[i]] %*% Q2[[j]] %*% Q1[[i]] (James and Wilkinson, 1971). An eigenvalue is regarded as zero if it is less than daeTolerance, which is initially set to .Machine$double.eps ^ 0.5 (about 1.5E-08). The function set.daeTolerance can be used to change daeTolerance.

However, a warning occurs if any pair of Q2 projectors (say Q2[[j]] and Q2[[k]]) do not have adjusted orthogonality with respect to any Q1 projector (say Q1[[i]]), because they are partially aliased. That is, if Q2[[j]] %*% Q1[[i]] %*% Q2[[k]] is nonzero for any pair of different Q2 projectors and any Q1 projector. When it is nonzero, the projector for the later term in the list of projectors is orthogonalized to the projector that is earlier in the list. A list of such projectors is returned in the aliasing component of the p2canon.object.

Value

A p2canon.object.

Author(s)

Chris Brien
projs.combine.p2canon

References


See Also

*summary.p2canon, efficiencies.p2canon, projs.combine.p2canon, pstructure, proj2.efficiency, proj2.combine, proj2.eigen, efficiency.criteria* in package *dae, eigen, projector* for further information about this class.

Examples

```r
## PBIBD(2) from p. 379 of Cochran and Cox (1957) Experimental Designs.
## 2nd edn Wiley, New York
PBIBD2.unit <- list(Block = 6, Unit = 4)
PBIBD2.nest <- list(Unit = "Block")
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 <- designRandomize(allocated = trt,
                               recipient = PBIBD2.unit,
                               nested.recipients = PBIBD2.nest)

##obtain projectors using pstructure
unit.struct <- pstructure(~ Block/Unit, data = PBIBD2.lay)
trt.struct <- pstructure(~ trt, data = PBIBD2.lay)

##obtain combined decomposition and summarize
unit.trt.p2canon <- projs.2canon(unit.struct$Q, trt.struct$Q)
summary(unit.trt.p2canon)
```

---

**projs.combine.p2canon**  Extract, from a p2canon object, the projectors that give the combined canonical decomposition

**Description**

Extracts, from a p2canon object obtained using *projs.2canon*, the projectors that give the combined canonical decomposition of two sets of projectors (Brien and Bailey, 2009).

**Usage**

```r
projs.combine.p2canon(object)
```

**Arguments**

- **object**  A list of class p2canon produced by *projs.2canon*. 
Value

A list, each of whose components is a projector in the decomposition.

Author(s)

Chris Brien

References


See Also

`projs.2canon`, `proj2.eigen`, `proj2.combine` in package `dae`.

`projector` for further information about this class.

Examples

```r
## PBIBD(2) from p. 379 of Cochran and Cox (1957) Experimental Designs.
## 2nd edn Wiley, New York
PBIBD2.unit <- list(Block = 6, Unit = 4)
PBIBD2.nest <- list(Unit = "Block")
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 <- designRandomize(allocated = trt,
recipient = PBIBD2.unit,
nested.recipients = PBIBD2.nest)

## obtain sets of projectors
unit.struct <- pstructure(~ Block/Unit, data = PBIBD2.lay)
trt.struct <- pstructure(~ trt, data = PBIBD2.lay)

## obtain combined decomposition
unit.trt.p2canon <- projs.2canon(unit.struct$Q, trt.struct$Q)
UcombineT <- projs.combine.p2canon(unit.trt.p2canon)
```

**pstructure.formula**

*Takes a formula and constructs a pstructure.object that includes the orthogonalized projectors for the terms in a formula*

Description

Constructs a `pstructure.object` that includes a set of mutually orthogonal projectors, one for each term in the formula. These are used to specify a structure, or an orthogonal decomposition of the data space. There are three methods available for orthogonalizing the projectors corresponding to the terms in the formula: differencing, eigenmethods or the default hybrid method.

It is possible to use this function to find out what sources are associated with the terms in a model and to determine the marginality between terms in the model. The marginality matrix can be saved.
Usage

```r
## S3 method for class 'formula'
pstructure(formula, keep.order = TRUE, grandMean = FALSE,
            orthogonalize = "hybrid", labels = "sources",
            marginality = NULL, check.marginality = TRUE,
            omit.projectors = FALSE,
            which.criteria = c("aEfficiency","eEfficiency","order"),
            aliasing.print = TRUE, data = NULL, ...)
```

Arguments

- **formula**: An object of class `formula` from which the terms will be obtained.
- **keep.order**: A logical indicating whether the terms should keep their position in the expanded formula projector, or reordered so that main effects precede two-factor interactions, which precede three-factor interactions and so on.
- **grandMean**: A logical indicating whether the projector for the grand mean is to be included in the set produced.
- **orthogonalize**: A character vector indicating the method for orthogonalizing a projector to those for terms that occurred previously in the formula. Three options are available: hybrid; differencing; eigenmethods. The hybrid option is the most general and uses the relationships between the projection operators for the terms in the formula to decide which projectors to subtract and which to orthogonalize using eigenmethods. The differencing option subtructs, from the current projector, those previously orthogonalized projectors for terms whose factors are a subset of the current projector's factors. The eigemethods option recursively orthogonalizes the projectors using an eigenanalysis of each projector with previously orthogonalized projectors.
- **labels**: A character nominating the type of labels to be used in labelling the projectors, and which will be used also in the output tables, such the tables of the aliasing in the structure. The two alternatives are terms and sources. Terms have all factors/variables in it separated by colons (:). Sources have factors/variables in them that represent interactions separated by hashes (#); if some factors are nested within others, the nesting factors are surrounded by square brackets ([ and ]) and separated by colons (:). If some generalized, or combined, factors have no marginal terms, the constituent factors are separated by colons (:) and if they interact with other factors in the source they will be parenthesized.
- **marginality**: A square matrix that can be used to supply the marginality matrix when it is desired to overwrite the calculated marginality matrix or when it is not being calculated. It should consist of zeroes and ones that gives the marginalites of the terms in the formula. It must have the row and column names set to the terms from the expanded formula, including being in the same order as these terms. The entry in the ith row and jth column will be one if the ith term is marginal to the jth term i.e. the column space of the ith term is a subspace of that for the jth term and so the source for the jth term is to be made orthogonal to that for the ith term. Otherwise, the entries are zero. A row and column should not be included for the grand mean even if grandMean is TRUE.
check.marginality

A logical indicating whether the marginality matrix, when it is supplied, is to be checked against that computed by `pstructure.formula`. It is ignored when orthogonalize is set to eigenmethods.

omit.projectors

A logical, which, if TRUE, results in the projectors in the Q of the pstructure.object being replaced by their degrees of freedom. These will be the degrees of freedom of the sources. This option is included a device for saving storage when the projectors are not required for further analysis.

which.criteria

A character vector nominating the efficiency criteria to be included in the summary of aliasing between terms. It can be none, all or some combination of aefficiency, mefficiency, sefficiency, xefficiency, order and dforthog – for details see `efficiency.criteria`. If none, no summary is printed.

aliasing.print

A logical indicating whether the aliasing between sources within the structure is to be printed.

data

A data frame contains the values of the factors and variables that occur in `formula`.

... further arguments passed to `terms`.

Details

Firstly, the primary projector \(X(X'X)^{-}X'\), where X is the design matrix for the term, is calculated for each term. Then each projector is made orthogonal to terms aliased with it, either by differencing, eigenmethods or the default hybrid method.

Differencing relies on comparing the factors involved in two terms, one previous to the other, to identify whether to subtract the orthogonalized projector for the previous term from the primary projector of the other. It does so if factors/variables for the previous term are a subset of the factors/variables for the other term. This relies on ensuring that all projectors whose factors/variables are a subset of the current projector occur before it in the expanded formula. It is checked that the set of matrices are mutually orthogonal. If they are not then a warning is given. It may happen that differencing does not produce a projector, in which case eigenmethods must be used.

Eigenmethods forces each projector to be orthogonal to all terms previous to it in the expanded formula. It uses equation 4.10 of James and Wilkinson (1971), which involves calculating the canonical efficiency factors for pairs of primary projectors. It produces a table of efficiency criteria for partially aliased terms. Again, the order of terms is crucial. This method has the disadvantage that the marginality of terms is not determined and so sources names are set to be the same as the term names.

The hybrid method is the most general and uses the relationships between the projection operators for the terms in the formula to decide which projectors to subtract and which to orthogonalize using eigenmethods. If \(Q_i\) and \(Q_j\) are two projectors for two different terms, with \(i < j\), then

1. if \(Q_j Q_i \neq 0\) then have to orthogonalize \(Q_j\) to \(Q_i\).
2. if \(Q_j Q_i = Q_j\) then, if \(Q_i = Q_j\), they are equal and \(Q_j\) will be removed from the list of terms; otherwise they are marginal and \(Q_i\) is subtracted from \(Q_j\).
3. If have to orthogonalize and $Q_jQ_i = Q_i$ then $Q_j$ is aliased with previous terms and will be removed from the list of terms; otherwise $Q_i$ is partially aliased with $Q_j$ and $Q_j$ is orthogonalized to $Q_i$ using eigenmethods.

Again, the order of terms is crucial in this process.

Value

A pstructure.object.

Author(s)

Chris Brien

References


See Also

proj2.efficiency, proj2.combine, proj2.eigen, projs.2canon in package dae, eigen.

projector for further information about this class.

Examples

```r
## PBIBD(2) from p. 379 of Cochran and Cox (1957) Experimental Designs.
## 2nd edn Wiley, New York
PBIBD2.unit <- list(Block = 6, Unit = 4)
PBIBD2.nest <- list(Unit = "Block")
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 <- designRandomize(allocated = trt,
                              recipient = PBIBD2.unit,
                              nested.recipients = PBIBD2.nest)
## manually obtain projectors for units
Q.G <- projector(matrix(1, nrow=24, ncol=24)/24)
Q.B <- projector(fac.meanop(PBIBD2.lay$Block) - Q.G)
Q.BP <- projector(diag(1, nrow=24) - Q.B - Q.G)
## manually obtain projector for trt
Q.T <- projector(fac.meanop(PBIBD2.lay$trt) - Q.G)

## compute intrablock efficiency criteria
effic <- proj2.efficiency(Q.BP, Q.T)
effic
efficiency.criteria(effic)

## obtain projectors using pstructure.formula
unit.struct <- pstructure(~ Block/Unit, data = PBIBD2.lay)
trt.struct <- pstructure(~ trt, data = PBIBD2.lay)
```
pstructure.object

Description

An object of class pstructure that contains information derived from a formula using pstructure.formula. It also inherits from class list.

Value

A list of class pstructure with the following components:

1. Q: a list with a component of class projector, being the orthogonalized projector for each non-aliased term/source in the formula; if grandMean is TRUE in the call to pstructure.formula then it also includes the projector for it;

2. terms: a character vector with the non-aliased term names; if grandMean is TRUE in the call to pstructure.formula then the first term will be "Mean";

3. sources: a character vector with the non-aliased source names;

4. marginality: a matrix of zeroes and ones with the same number of rows and columns as number of non-aliased terms, excluding the term for the grand mean even when grandMean is TRUE; the row names and column names are the elements terms, excluding "Mean"; the entry in the ith row and jth column will be one if the ith term is marginal to the jth term i.e. the column space of the ith term is a subspace of that for the jth term and so the source for the jth term will have been made orthogonal to that for the ith term; otherwise, the entries are zero.

5. aliasing: a data.frame containing the information about the (partial) aliasing between the sources in the formula. The columns are:
   • Source: the source names, or associated term name, for those that are (partially) aliased with previous sources;
   • df: the remaining degrees of freedom for the source;
   • Alias: the source with which the current entry is (partially) aliased;
   • efficiency criteria: a set of columns for the complete set of criteria calculated by efficiency.criteria.

The object has the attribute labels, which is set to "terms" or "sources" according to which of these label the projectors.

# obtain combined decomposition and summarize
unit.trt.p2canon <- projs.2canon(unit.struct$Q, trt.struct$Q)
summary(unit.trt.p2canon, which = c("aeff","eeff","order"))
**qqyeffects**

**Author(s)**
Chris Brien

**See Also**
- `pstructure.formula`
- `projector`

---

**Description**

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

**Usage**

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

**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.
- `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 output.

**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

```r
## analysis of 2^4 factorial experiment from Table 10.6 of Box, Hunter and
## 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(...)

Arguments

...  
Arguments passed to residuals.aovlist.

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

```r
## 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 variance
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

```r
## 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

```r
## 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 variance
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 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 $n$ 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 $\text{mean}$. 

Value

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

Author(s)

Chris Brien

References


See Also

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

Examples

```r
## 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)
```

Description

The data is from an experiment involved two phases. In the field phase a viticultural experiment was conducted to investigate the differences between 4 types of trellising and 2 methods of pruning. The design was a split-plot design in which the trellis types were assigned to the main plots using two adjacent Youden squares of 3 rows and 4 columns. Each main plot was split into two subplots (or halfplots) and the methods of pruning assigned at random independently to the two halfplots in each main plot. The produce of each halfplot was made into a wine so that there were 24 wines altogether.

The second phase was an evaluation phase in which the produce from the halfplots was evaluated by 6 judges all of whom took part in 24 sittings. In the first 12 sittings the judges evaluated the wines made from the halfplots of one square; the final 12 sittings were to evaluate the wines from the other square. At each sitting, each judge assessed two glasses of wine from each of the halfplots of one of the main plots. The main plots allocated to the judges at each sitting were determined as follows. For the allocation of rows, each occasion was subdivided into 3 intervals of 4 consecutive sittings.
During each interval, each judge examined plots from one particular row, these being determined using two 3x3 Latin squares for each occasion, one for judges 1-3 and the other for judges 4-6. At each sitting judges 1-3 examined wines from one particular column and judges 4-6 examined wines from another column. The columns were randomized to the 2 sets of judges x 3 intervals x 4 sittings using duplicates of a balanced incomplete block design for v=4 and k=2 that were latinized. This balanced incomplete block design consists of three sets of 2 blocks, each set containing the 4 "treatments". For each interval, a different set of 2 blocks was taken and each block assigned to two sittings, but with the columns within the block placed in reverse order in one sitting compared to the other sitting. Thus, in each interval, a judge would evaluate a wine from each of the 4 columns.

The data.frame contains the following factors, in the order give: Occasion, Judges, Interval, Sittings, Position, Squares, Rows, Columns, Halfplot, Trellis, Method. They are followed by the simulated response variable Score.

The scores are ordered so that the factors Occasion, Judges, Interval, Sittings and Position are in standard order; the remaining factors are in randomized order.

See also the vignette daeDesignNotes.pdf.

set.daeTolerance

A function that sets the values such that, in dae functions, values less than it are considered to be zero. The values are stored in a vector named daeTolerance in the daeEnv environment. The vector is of length two and, initially, both values are set to .Machine$double.eps ^ 0.5 (about 1.5E-08). One value is named element.tol and is used for elements of matrices; the second is named element.eigen and is used for eigenvalues and quantities based on them, such as efficiency factors.

Usage

set.daeTolerance(element.tol=NULL, eigen.tol=NULL)
Arguments

- `element.tol`  The value to which the first element of the daeTolerance vector is to be set. If more than one value is supplied, only the first value is used.
- `eigen.tol`  The value to which the second element of the daeTolerance vector is to be set. If more than one value is supplied, only the first value is used.

Value

The vector daeTolerance is returned invisibly.

Author(s)

Chris Brien

See Also

`get.daeTolerance`.

Examples

```r
## set daeTolerance.
set.daeTolerance(1E-04, 1E-08)
```

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**

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

**Format**

A data.frame containing 36 observations of 8 variables.

**Source**


---

**strength**

Generate paper strength values

**Description**

Generates paper strength values for an experiment with different temperatures.

**Usage**

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

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.
## eradicate the unrandomized version of temperature

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 <- designRandomize(allocated = temperature,
                          recipient = list(runs = 12),
                          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)
summary.p2canon

**Summarize a canonical analysis of the relationships between two sets of projectors**

**Description**

Produces a summary of the efficiency criteria computed from the canonical efficiency factors for the joint decomposition of two sets of projectors (Brien and Bailey, 2009) obtained using `projs.2canon`. It takes the form of a decomposition or skeleton ANOVA table.

**Usage**

```r
## S3 method for class 'p2canon'
summary(object, which.criteria = c("aefficiency", "eefficiency", "order"), ...)
```

**Arguments**

- `object` A list of class `p2canon` produced by `projs.2canon`.
- `which.criteria` A character vector nominating the efficiency criteria to be included in the summary. It can be none, all or some combination of `aefficiency`, `mefficiency`, `sefficiency`, `eefficiency`, `xefficiency`, `order` and `dforthog` – for details see `efficiency.criteria`.
- `...` Further arguments affecting the summary produced.

**Value**

An object of classes `summary.p2canon` and `data.frame`, whose rows correspond to the pairs of projectors, one from the `Q1` argument and the other from the `Q2` argument from `projs.2canon`; only pairs with non-zero efficiency factors are included. In addition, a line is included for each nonzero Residual `Q1` projector.

**Author(s)**

Chris Brien

**References**


**See Also**

- `projs.2canon`, `proj2.efficiency`, `efficiency.criteria`, `proj2.combine`, `proj2.eigen`, `pstructure`, `print.summary.p2canon` 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(Block = 6, Unit = 4)
PBIBD2.nest <- list(Unit = "Block")
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 <- designRandomize(allocated = trt,
recipient = PBIBD2.unit,
nested.recipients = PBIBD2.nest)

##obtain projectors using pstructure
unit.struct <- pstructure(~ Block/Unit, data = PBIBD2.lay)
trt.struct <- pstructure(~ trt, data = PBIBD2.lay)

##obtain combined decomposition and summarize
unit.trt.p2canon <- projs.2canon(unit.struct$Q, trt.struct$Q)
summary(unit.trt.p2canon)

---

**summary.pcanon**

*Summarizes the anatomy of a design, being the decomposition of the sample space based on its canonical analysis, as produced by designAnatomy*

Description

Gives the anatomy of a design in a table; it summarizes the joint decomposition of two or more sets of projectors (Brien and Bailey, 2009) obtained using designAnatomy. It includes the efficiency criteria computed from the canonical efficiency factors for the joint decomposition. The labels in the table may be Terms or Sources. The terms are those that would be included in a mixed model for an experiment based on the design. The sources are the orthogonal subspaces, derived from the terms, that make up the decomposition and the degrees of freedom and efficiency factors relate to these subspaces. The table displays how the information for the different sources allowed for in the design are related. For more information about the notation used for sources see the labels argument of designAnatomy.

It is possible to supply an object that is a pcanon.object produced in versions prior to 3.0-0 using projs.cannon.

Usage

## S3 method for class 'pcanon'
summary(object, labels.swap = FALSE,
which.criteria = c("aefficiency", "eefficiency", "order"), ...)

Arguments

- **object** A pcanon.object.
labels.swap  A logical indicating whether to swap between "sources" and 'terms' in the output. The default is established by the labels argument of designAnatomy and projs.canon.

which.criteria  A character vector nominating the efficiency criteria to be included in the summary. It can be none, all or some combination of aefficiency, mefficiency, sefficiency, eefficiency, xefficiency, order and dforthog – for details see efficiency.criteria. If there is only one formula, this argument is ignored.

... further arguments affecting the summary produced.

Value
An object of class summary.pcanon that is a list with the two components decomp and aliasing.

The component decomp is a data.frame whose rows correspond to subspaces in the decomposition for a design. It has the following attributes: (i) title that is the title for printing with the decomposition table; (ii) ntiers that is equal to the number of tiers; (iii) orthogonal that is TRUE if the design is orthogonal; (iv) labels that is either "terms" or "sources" depending on the labels that have resulted from the setting of label.swap.

The component aliasing is a data.frame that is also of class aliasing. It contains information about the aliasing between terms that are derived from the same formula and has the attribute title that is the title to be printed with the aliasing table.

However, if the object supplied is a pcanon.object produced with versions prior to 3.0-0 using projs.canon, the value is a data.frame, instead of a list, that has the same attributes as the decomp component of the summary.pcanon object now produced, except that labels is always set to "terms".

Author(s)
Chris Brien

References

See Also
designAnatomy, designAnatomy, pstructure, efficiency.criteria, proj2.combine, proj2.efficiency, proj2.eigen, print.summary.pcanon 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(Block = 6, Unit = 4)
PBIBD2.nest <- list(Unit = "Block")
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))
tukey.1df <- designRandomize(allocated = trt,
    recipient = PBIBD2.unit,
    nested.recipients = PBIBD2.nest)

# obtain combined decomposition and summarize
unit.trt.canon <- designAnatomy(list(unit=~ Block/Unit, trt=~ trt),
    data = PBIBD2.lay)
summary(unit.trt.canon, which = c("aeff","eef","order"))
summary(unit.trt.canon, which = c("aeff","eef","order"), labels.swap = TRUE)

## 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.
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.
**Zncsspline**

*Calculates the design matrix for fitting the random component of a natural cubic smoothing spline*

**Description**

Calculates the design matrix, $Z$, of the random effects for a natural cubic smoothing spline as described by Verbyla et al., (1999). An initial design matrix, $\Delta \Delta^{-1} \Delta$, based on the knot points is computed. It can then be post multiplied by the power of the tri-diagonal matrix $G_s$ that is proportional to the variance matrix of the random spline effects. If the power is set to 0.5 then the random spline effects based on the resulting $Z$ matrix will be independent with variance $\sigma_s^2$.

**Usage**

`Zncsspline(knot.points, Gpower = 0, print = FALSE)`

**Arguments**

- **knot.points**: A numeric giving the values of the knot points to be used in fitting the spline. These must be ordered in increasing order.
- **Gpower**: A numeric giving the power of the tri-diagonal matrix $G_s$, from which the variance matrix of the random spline effects is calculated. That the initial design matrix is to be the value of the variance component for the random spline effects. The smoothing parameter is then the inverse of the ratio of this component to the residual variance.
- **print**: A logical indicating whether to print the $\Delta$ and $G_s$ matrices.

**Examples**

```r
## 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)
```
Value

A matrix containing the design matrix.

Author(s)

Chris Brien

References


See Also

mat.ncssvar.

Examples

```
Z <- Zncsspline(knot.points = 1:10, Gpower = 0.5)
```
Index

*Topic **aplot**
  designGGPlot, 27
  designPlotlabels, 33
  interaction.ABC.plot, 64
*Topic **array**
  as.data.frame.pstructure, 9
  correct.degfree, 14
  decomp.relate, 16
  degfree, 18
  designAnatomy, 21
  designLatinSqrSys, 30
  designTwoPhaseAnatomies, 38
  efficiencies, 41
  efficiency.criteria, 43
  elements, 44
  fac.ar1mat, 47
  fac.meanop, 53
  fac.sumop, 58
  fac.vcmat, 59
  is.projector, 67
  marginality, 68
  mat.ar1, 70
  mat.ar2, 70
  mat.ar3, 71
  mat.arma, 73
  mat.banded, 74
  mat.dirprod, 75
  mat.dirmum, 76
  mat.exp, 77
  mat.gau, 77
  mat.I, 78
  mat.J, 79
  mat.ma1, 80
  mat.ma2, 80
  mat.ncssvar, 81
  mat.sar, 82
  mat.sar2, 83
  mat.Vpred, 84
  mat.Vpredicts, 86
  p2canon.object, 93
  pcanon.object, 94
  print.aliases, 96
  print.projection, 97
  print.pstructure, 98
  proj2.combine, 101
  proj2.efficiency, 102
  proj2.eigen, 104
  projector, 105
  projector-class, 106
  projs.2canon, 108
  projs.combine.p2canon, 109
  pstructure.formula, 110
  pstructure.object, 114
  show-methods, 121
  summary.p2canon, 124
  summary.pcanon, 125
  Zncsspline, 129
*Topic **asreml**
  daeTips, 16
*Topic **classes**
  projector-class, 106
*Topic **datagen**
  designRandomize, 35
  fac.gen, 50
  rmvnorm, 118
  strength, 122
*Topic **datasets**
  ABC.Interact.dat, 8
  BIBD.Wheat.dat, 11
  Cabinet1.des, 13
  Casuarina.dat, 14
  Exp249.munit.des, 45
  Fac4Proc.dat, 60
  LatticeSquare_t49.des, 68
  McIntyreTMV.dat, 89
  Oats.dat, 92
  Sensory3Phase.dat, 119
  SPLGrass.dat, 122

131
*Topic design
  blockboundaryPlot, 11
decomp.relate, 16
designAnmeasures, 19
designAnatomy, 21
designBlocksGGPlot, 24
designGGPlot, 27
designLatinSqrSys, 30
designPlot, 30
designPlotlabels, 33
designRandomize, 35
designTwophaseAnatomies, 38
detect.diff, 40
efficiencies, 41
efficiency.criteria, 43
fac.gen, 50
fac.match, 52
interaction.ABC.plot, 64
marginality, 68
mat.ncssvar, 81
mat.Vpred, 84
mat.Vpredicts, 86
no.reps, 91
p2canon.object, 93
pcanon.object, 94
power.exp, 95
print.summary.p2canon, 99
print.summary.pcanon, 100
proj2.combine, 101
proj2.efficiency, 102
proj2.eigen, 104
projs.2canon, 108
projs.combine.p2canon, 109
pstructure.formula, 110
pstructure.object, 114
qqyeffects, 115
strength, 122
summary.p2canon, 124
summary.pcanon, 125
yates.effects, 128
Zncsspline, 129

*Topic factor
  as.numfac, 10
designRandomize, 35
fac.combine, 48
fac.divide, 49
fac.gen, 50
fac.match, 52
fac.multinested, 54
fac.nested, 56
fac.recode, 57
mpone, 90

*Topic hplot
  designGGPlot, 27
designPlotlabels, 33
interaction.ABC.plot, 64
qqyeffects, 115

*Topic htest
  fitted.aovlist, 60
fitted.errors, 62
qqyeffects, 115
resid.errors, 116
residuals.aovlist, 117
tukey.1df, 127
yates.effects, 128

*Topic iplot
  qqyeffects, 115

*Topic manip
  as.numfac, 10
  elements, 44
  extab, 46
  fac.combine, 48
  fac.divide, 49
  fac.multinested, 54
  fac.nested, 56
  fac.recode, 57
get.daeTolerance, 63
harmonic.mean, 64
is.allzero, 66
mpone, 90
set.daeTolerance, 120

*Topic methods
  fitted.aovlist, 60
residuals.aovlist, 117
show-methods, 121

*Topic models
  fitted.aovlist, 60
fitted.errors, 62
resid.errors, 116
residuals.aovlist, 117
tukey.1df, 127

*Topic package
  dae-package, 4

*Topic plot
  blockboundaryPlot, 11
designBlocksGGPlot, 24
designPlot, 30

+Topic projector
  as.data.frame.pstructure, 9
correct.degfree, 14
decomp.relate, 16
degfree, 18
designFactor, 21
designTwoPhaseAnatomies, 38
efficiencies, 41
efficiency.criteria, 43
fac.meanop, 53
fac.sumop, 58
get.daeTolerance, 63
is.projector, 67
marginality, 68
p2canon.object, 93
pcanon.object, 94
print.aliasing, 96
print.projector, 97
print.pstructure, 98
print.summary.p2canon, 99
print.summary.pcanon, 100
proj2.combine, 101
proj2.efficiency, 102
proj2.eigen, 104
projector, 105
projector-class, 106
projs.2canon, 108
projs.combine.p2canon, 109
pstructure.formula, 110
pstructure.object, 114
set.daeTolerance, 120
show-methods, 121
summary.p2canon, 124
summary.pcanon, 125

ABC.Interact.dat, 4, 8
Ameasures (dae-deprecated), 15
aov, 61, 62, 115–117, 127–129
array, 107
as.data.frame, 10
as.data.frame.pstructure, 8, 9
as.numeric, 10
as.numfac, 5, 10, 58

BIBD.Wheat.dat, 4, 11
blockboundary.plot (dae-deprecated), 15
blockboundaryPlot, 5, 11, 33

Cabinet1.des, 13
Casuarina.dat, 4, 14
character, 9, 12, 22, 23, 25, 28, 31, 32, 35,
  36, 38, 39, 83, 87, 94, 111, 114, 126
chol, 119
coerce.projector, matrix-method
  (projector-class), 106
correct.degfree, 7, 14, 18, 54, 67, 106, 107
dae (dae-package), 4
dae-deprecated, 15
dae-package, 4
daetips, 16
data.frame, 9, 21, 27, 34–36, 38, 49–51, 55,
  65, 87, 94, 96, 114
decomp.relate, 8, 16, 102
degfree, 7, 15, 18, 54, 106, 107
degfree<- (degfree), 18
design.plot (dae-deprecated), 15
designAnatomy, 5, 7, 20, 21, 30, 33, 36,
  38–42, 68, 69, 93, 94, 125, 126
designBlocksGGPlot, 5, 24
designGGPlot, 5, 24, 26, 27
designLatinSqrSys, 5, 23, 30, 33, 36
designPlot, 5, 11, 12, 23, 29, 30, 34, 36
designPlotLabels, 5, 33, 33
designRandomize, 5, 23, 30, 33, 35
designTwoPhaseAnatomies, 5, 38
detect.diff, 6, 40, 92, 95
efficiencies, 41
efficiencies.p2canon, 7, 109
efficiencies.pcanon, 7, 23, 40
efficiency.criteria, 8, 22, 23, 39, 40, 43,
  93, 96, 98, 103, 109, 112, 114, 124, 126
eigen, 17, 24, 40, 42, 43, 69, 103, 104, 109,
  113, 124, 126
elements, 6, 44
Exp249.munit.des, 4, 45
extab, 8, 46

fac.ar1mat, 6, 47, 60, 119
fac.combine, 5, 29, 34, 48, 50, 51, 53, 54, 58,
  59, 64, 65
fac.divide, 5, 49, 49
INDEX

fac.recode, fac.multinested, fac.gen
134
INDEX
mat.exp, mat.ar3, mat.ar2, mat.ar1
marginality, interaction.ABC.plot
integer, geom_text, get.daeTolerance, ggplot
harmonic.mean

integer, 31
interaction.plot, interaction.plot
is.allzero, is.projector, LatticeSquare_t49.des
list, logical
marginality, marginality
mat.ar1, mat.ar2, mat.ar3
mat arma, mat.banded, mat.dirprod, mat.exp, mat.gau
mat.I, mat.J, mat.ml, mat.mal, mat.ma2, mat.ncssvar, mat.sar, mat.sar2, mat.Vpred
mat.Vpredicts, match, matrices
meanop, mpone
no.reps, numeric

Oats.dat, p2canon.object, par
pcanon.object, polygon, power.exp, print
print.projector-method
printaliasing, print.default, print.projector, print.pstructure
print.summary.p2canon, print.summary.pcanon
proj2 combine, proj2.decomp
proj2.efficiency, proj2.eigen
proj2.ops

134
INDEX

projector, 7, 14, 15, 18, 24, 40, 42, 43, 53, 54, 67, 69, 85, 87, 97, 102–105, 105, 106–113, 115, 121, 124, 126
projector-class, 7, 106
projs.2canon, 7, 109, 109
projs.2canon, 7
projs.2canon, 7
projs.cannon, 126
projs.cannon (dae-deprecated), 15
projs.combine.p2canon, 7
projs.combine.p2canon, 7
pstructure, 107
pstructure, 107
pstructure, 107
pstructure, 107
pstructure (dae-deprecated), 15
pstructure, 23, 40, 42, 69, 109, 124, 126
pstructure (pstructure.formula), 110
pstructure.formula, 8, 22, 68, 69, 94, 110, 112, 114, 115
pstructure.object, 5, 9, 68, 69, 94, 98, 110, 113, 114

qqnorm, 116
qqyeffects, 6, 115, 129

relevel, 58, 90
resid.errors, 6, 61, 62, 116, 117, 128
residuals, 116, 117
residuals (residuals.aovlist), 117
residuals.aovlist, 6, 116, 117
rmvnorm, 8, 118
rnorm, 119

Sensory3Phase.dat, 4, 119
Sensory3PhaseShort.dat, 4
Sensory3PhaseShort.dat
(Sensory3Phase.dat), 119
set.daeTolerance, 8, 14, 17, 18, 63, 64, 66, 67, 88, 101, 103–105, 108, 120
show, 96–98
show, ANY-method (show-methods), 121
show, classRepresentation-method
(show-methods), 121
show, genericFunction-method
(show-methods), 121
show, MethodDefinition-method
(show-methods), 121
show, MethodSelectionReport-method
(show-methods), 121
show, MethodWithNext-method
(show-methods), 121
show, ObjectsWithPackage-method
(show-methods), 121
show, oldClass-method (show-methods), 121

show, projector-method (show-methods), 121
show, signature-method (show-methods), 121
show, traceable-method (show-methods), 121
show-methods, 8, 121
SPLGrass.dat, 4, 122
strength, 6, 122
structure, 107
summary, p2canon-method
(summary.p2canon), 124
summary, pcanon-method (summary.pcanon), 125
summary, p2canon, 7, 99, 109, 124
summary.pcanon, 6, 7, 21, 23, 38, 40, 42, 69, 100, 125

text, 31
tukey.1df, 6, 61, 62, 116, 117, 127

vector, 52, 107, 119

yates.effects, 6, 116, 128

Zncsspline, 7, 82, 129