Package 'AlgDesign'

July 21, 2025

Version 1.2.1.2

Title Algorithmic Experimental Design

Maintainer Jerome Braun <jvbraun.statistics@gmail.com>

Description Algorithmic experimental designs. Calculates exact and approximate theory experimental designs for D,A, and I criteria. Very large designs may be created. Experimental designs may be blocked or blocked designs created from a candidate list, using several criteria. The blocking can be done when whole and within plot factors interact.

License GPL (>= 2)

URL https://github.com/jvbraun/AlgDesign

Repository CRAN

Date/Publication 2025-04-06 08:00:12 UTC

NeedsCompilation yes

Author Bob Wheeler [aut],
Jerome Braun [cre]

Contents

`CCTable11.1a` *Cochran and Cox design*

Description

A blocked experimental design from Cochran and Cox (1957).

A Partially balanced incomplete block in 3 reps, and 27 blocks.

Usage

```
data(CCTable11.1a)
```

Source

Cochran, W.G. and Cox, G.M. (1957). *Experimental designs*. Wiley, N.Y.

`efficient.rounding` *Efficient Rounding*

Description

A vector of proportions is efficiently rounded to integers.

Usage

```
efficient.rounding(proportions,n,random=TRUE)
```

Arguments

`proportions`	A vector of proportions.
`n`	The sum of the resulting integers.
`random`	If TRUE, ties will be broken at random, otherwise the first of the tied values will be used.

Details

This function implements an efficient rounding procedure to round approximate theory designs into replicated integer approximations.

Value

A vector of replications summing to n.

Author(s)

Bob Wheeler <bwheelerg@gmail.com>

Please cite this program as follows:

Wheeler, R.E. (2004). efficient.rounding. *AlgDesign*. The R project for statistical computing https://www.r-project.org/

References

Pulkesheim, F. and Rieder, S. (1992). Efficient rounding of approximate designs. Biometrika. 79-4. 763-770.

`eval.blockdesign` *Evaluates a blocked design.*

Description

A blocked design is evaluated.

Usage

```
eval.blockdesign(frml,design,blocksizes,rho=1,confounding=FALSE,center=FALSE)
```

Arguments

`frml`	The formula used to create the blocked design.
`design`	The blocked design, which may be the design output by optBlock().
`blocksizes`	A vector of blocksizes for the design.
`rho`	A vector, giving the ratios of whole to within variance components.
`confounding`	If confounding=TRUE, the confounding matrix will be output.
`center`	If TRUE, numeric variables will be centered before frml is applied.

Value

`confounding`	A matrix based on the design matrix in which the within block variables have been centered about their block means. The columns of this matrix which give the regression coefficients of each variable regressed on the others. If C is the confounding matrix, then $-XC$ is a matrix of residuals of the variables regressed on the other variables.
`determinant.all.terms.within.terms.centered`	$\|M\|^{1/k}$, where $M = X^T X/N$ and X is the model expanded $N \times k$ design matrix in which the within block variables have been centered about the grand mean.

`within.block.efficiencies`
: The determinant criterion blocking efficiencies for the range of rho's input. A high efficiency indicates that there is little intrablock information to be recovered in the analysis.

`block.centered.properties`
: A matrix with four rows. The columns correspond to constant, whole block terms and within block terms:

1. The degrees of freedom for terms in the expanded model.
2. The determinant of the block centered within block terms.
3. The geometric mean of the block centered variances.
4. The geometric mean of the ratio of centered to block centered variances.

Author(s)

Bob Wheeler <bwheelerg@gmail.com>

Please cite this program as follows:

Wheeler, R.E. (2004). eval.blockdesign. *AlgDesign*. The R project for statistical computing https://www.r-project.org/

---

eval.design *Evaluates a design.*

Description

A design is evaluated.

Usage

```
eval.design(frml,design,confounding=FALSE,variances=TRUE,center=FALSE,X=NULL)
```

Arguments

`frml`	The formula used to create the design.
`design`	The design, which may be the design part of the output of optFederov().
`confounding`	If confounding=TRUE, the confounding patterns will be shown.
`variances`	If TRUE, the variances each term will be output.
`center`	If TRUE, numeric variables will be centered before frml is applied.
`X`	X is either the matrix describing the prediction space for I or for G, the the candidate set from which the design was chosen. They are often the same.

Value

`confounding` A matrix. The columns of which give the regression coefficients of each variable regressed on the others. If C is the confounding matrix, then $-ZC$ is a matrix of residuals of the variables regressed on the other variables.

`determinant` $|M|^{1/k}$, where $M = Z'Z/N$, and Z is the model expanded $N \times k$ design matrix.

`A` The average coefficient variance: $trace(Mi)/k$, where Mi is the inverse of M.

`I` The average prediction variance over X, which can be shown to be $trace((X'X * Mi)/N)$.

`Ge` The minimax normalized variance over X, expressed as an efficiency with respect to the optimal approximate theory design. It is defined as $k/max(d)$, where $max(d)$ is the maximum normalized variance over X – i.e. the max of $x'(Mi)x$, over all rows x' of X.

`Dea` A lower bound on `D` efficiency for approximate theory designs. It is equal to $exp(1 - 1/Ge)$.

`diagonality` The diagonality of the design, excluding the constant, if any. Diagonality is defined as $(|M_1|/\prod diag(M_1))^{1/k}$, where M_1 is M with first column and row deleted when there is a constant.

`gmean.variances`
The geometric mean of the coefficient variances.

Note

I, Ge and Dea are calculated only when X is input.

Author(s)

Bob Wheeler <bwheelerg@gmail.com>

Please cite this program as follows:

Wheeler, R.E. (2004). eval.design. *AlgDesign*. The R project for statistical computing https://www.r-project.org/

`expand.formula` *Expanding a formula*

Description

Formulas are expanded to accommodate special functions for continuous and mixture variables.

Usage

```
expand.formula(frml,varNames,const=TRUE,numerics=NULL)
```

Arguments

`frml`	A formula starting with ~ in the usual way.
`varNames`	A list of variable names to be used when a dot is used as shorthand for "all variables."
`const`	If FALSE, the constant will be suppressed.
`numerics`	A vector the same length as varNames, with TRUE for corresponding numeric variables. If missing, all variables will be assumed to be numeric.

Details

This function expands formulas to accommodate polynomial models for which R has minimal support. Assuming for illustration that there are three variables, A, B, and C, the following expressions may be used. In addition, a dot may be used to indicate that all variables in varNames are to be used.

All agruments to quad(), cubic(), and cubicS() must be numeric.

- . makes $A + B + C$
- $.^p$ makes $(A + B + C)^p$, where p is an integer
- quad(A,B,C) makes $(A + B + C)^2 + I(A^2) + I(B^2) + I(C^2)$
- cubic(A,B,C) makes $(A + B + C)^3 + I(A^2) + I(B^2) + I(C^2) + I(A^3) + I(B^3) + I(C^3)$
- cubicS(A,B,C) makes $(A + B + C)^3 + I(A * B * (A - B)) + I(A * C * (A - C)) + I(B * C * (B - C))$

The cubicS() function produces a non-singular representation of a cubic model, when the variables are mixture variables, that is when the rows of `data` sum to a constant value, usually 1.0. Because of the mixture constraint, models containing mixture variables should not have a constant term. The linear and quadratic models for mixture variables A, B, and C are given by $-1 + (A + B + C)$ and $-1 + (A + B + C)^2$ respectively. See Gorman and Hinman [1962] for details.

Value

An expanded formula is returned.

Note

expand.formula() is called by model.matrix() through the method call model.matix.formula(), thus one may use the above special functions with model.matrix().

Author(s)

Bob Wheeler <bwheelerg@gmail.com>

Please cite this program as follows:

Wheeler, R.E. (2004). expand.formula. *AlgDesign*. The R project for statistical computing https://www.r-project.org/

References

Gorman, J.W. and Hinman, J.E. (1962). Simplex lattice designs for multicomponent systems. *Technometrics*. 4-4. 463-487.

`gen.factorial` *Generates a full factorial design*

Description

A full factorial design is generated.

Usage

```
gen.factorial(levels, nVars=0, center=TRUE, factors="none",varNames=NULL)
```

Arguments

`levels`	A vector of levels for the variables. May be an integer if nVars is specified.
`nVars`	The number of variables.
`center`	If TRUE, all non-factors will be centered.
`factors`	If "all", all variables are factors, otherwise a vector of the variable numbers of the variables that are to be factors.
`varNames`	The names of the variables.

Value

A factorial design in a data.frame.

Author(s)

Bob Wheeler <bwheelerg@gmail.com>

Please cite this program as follows:

Wheeler, R.E. (2004). gen.factorial. *AlgDesign*. The R project for statistical computing https://www.r-project.org/

Examples

```
dat<-gen.factorial(3,3)
dat<-gen.factorial(c(3,2,3))
dat<-gen.factorial(3,3,factors="all")
dat<-gen.factorial(3,3,varNames=c("A","B","C"))
```

gen.mixture *Generate mixture*

Description

Creates a candidate list of mixture variables.

Usage

```
gen.mixture(levels,vars)
```

Arguments

levels	An integer greater than 1. The number of levels of the mixture variables.
vars	Either the number of variables, or a character vector of variable names.

Details

Similar in function to gen.factorial(), but for mixture variables such that the rows sum to unity. For `levels=2`, the identity matrix is produced, so that each variable is either 0 or 1. For `levels=3`, rows are added to the identity containing two variables at 1/2 and the others at 0. etc.

In general, a mixture model of degree d will have the same number of terms as the candidate list generated by `gen.mixture` for `levels=d+1`, and this will be optimal.

Author(s)

Bob Wheeler <bwheelerg@gmail.com>

Please cite this program as follows:

Wheeler, R.E. (2004). gen.mixture. *AlgDesign*. The R project for statistical computing https://www.r-project.org/

GVTable1 *Goos Vandebroek Table 1*

Description

A blocked experimental design from Goos and Vandebroek (2003).

A matrix giving one whole plot and four within plot variables, in 21 blocks of 2. The same as TGTable5, but produced by Goos Vandoebroek algorithm.

Usage

```
data(GVTable1)
```

Source

Goos, P. and Vandebroek, M. (2003). D-optimal split-plot designs with given numbers and sizes of whole plots. *Technometrics.* 45-3. 235-245.

`GVTable3` *Goos Vandebroek Table 3*

Description

A blocked experimental design from Goos and Vandebroek (2003).

A matrix giving one whole and two within plot variables, 9 blocks of 21.

Usage

```
data(GVTable3)
```

Source

Goos, P. and Vandebroek, M. (2003). D-optimal split-plot designs with given numbers and sizes of whole plots. *Technometrics.* 45-3. 235-245.

`model.matrix.formula` *Builds a model matrix*

Description

Produces a model matrix using expand.formula()

Usage

```
## S3 method for class 'formula'
model.matrix(frml,data,...)
```

Arguments

`frml`	A formula.
`data`	A data.frame
`...`	Additional arguments passed to model.matrix.default()

Details

This is a method function signaled by a formula as the first argument. It causes the formula to be translated by expand.formula() before calling model.matrix.default().

Value

A matix.

Author(s)

Bob Wheeler <bwheelerg@gmail.com>

Please cite this program as follows:

Wheeler, R.E. (2004). model.matrix.formula. *AlgDesign*. The R project for statistical computing https://www.r-project.org/

---

optBlock *Optimal design blocking*

Description

Blocking of experimental designs using various criteria.

Usage

```
optBlock(frml,withinData,blocksizes,rows=NULL,wholeBlockData=NULL,center=FALSE,
nRepeats=5,criterion="D",args=FALSE)
```

Arguments

frml	This may be omitted if data is the fully model expanded candidate list. If present it should be a formula starting with ~ which describes the model using variable names from data. It may be ~. if all variables from data are to be used linearly. It may also be $.^p$, where p is an integer. In addition to the usual operators, quad(), cubic() and cubicS() may be used to expand variables from data into polynomial models.
withinData	A matrix or data.frame describing the variables. If the columns are not named, they will be assumed to have the names X1,X2, etc. Although data may be input as global variables used in frml, it is preferable to input it here. The number of rows in withinData must be at least as large as the sum of the number of terms plus the number of blocks.
blocksizes	A vector giving the block sizes for each block. The length of blocksizes specifies the number of blocks.
rows	Row numbers (not rownames) of withinData rows to be used as a starting blocked design.
wholeBlockData	A matrix or data.frame describing the whole block variables. The number of rows should equal the length of blocksizes. Each row should correspond to the settings of the whole block variable in a block.
center	If TRUE, the withinData and wholeBlockData will be centered.

nRepeats	The number of times the entire process is repeated.
criterion	"D":D-criterion: "OB","OBS": orthogonal blocks, unscaled and scaled; "Dp","Dpc": Product of block determinants, uncentered and centered.
args	If TRUE, the actual arguments to the function including the starting random number seed should be output.

Details

Cook and Nachtsheim (1989) developed a blocking algorithm using the D criterion, which not only finds blocks, but attempts to optimize the design itself. The package Dopt, converted by V.N. Venables, implements this technology – unfortunately there seem to be bugs in the FORTRAN code.

This function uses various criteria to block either pre-existing designs or to creates a new blocked design from a candidate list. There may be interactions between whole plot and within plot factors, and it may be used to produce multistratum blocked designs.

The algebraic expressions used here for the D criterion are quite different than those used by either Cook and Nachtshim or Goos and Vandebroek. They are described in *Comments on algorithmic design*, a paper accompanying this package.

Although the D criterion produces good designs for all blocks, the allocation for individual blocks can often be improved upon by the use of either the Dp or Dpc criterion, which optimize the product of the determinants of the individual blocks. The Dp criterion uses uncentered blocks, the Dpc uses centered blocks.

Blocking may also be done using the orthogonal blocking procedure of Nguyen. This comes in two flavors OB and OBS which differ in that for OBS, the columns of S are scaled by division with the sample variance of the data, thus deemphasizing squared and other large terms.

Blocking may be done when whole plot factors interact with within plot factors. Split plot experiments are sometimes of this nature. Goos and Vandebroek (2003) developed a procedure for this problem which algorithmically blocks using a candidate list and the ratio between the whole and within variances. Trinca and Gilmour (2001) give an algorithm which does not depend on the ratio of the variances. The present procedure assumes that the whole blocks and their factors, have been decided upon in advance. It makes no assumptions about the ratio between whole and within variances.

A vignette giving further details is availble. To access it, type

vignette("AlgDesign")

Value

D	$det(M)^{1/k}$, where $det(M)$ is the determinant of the normalized dispersion matrix M – i.e. $M = X'X/N$, where each row of X has had the appropriate block mean subtracted.
Dp (Dpc)	$\left(\prod_1^b (det(M_i))^{1/k}\right)^{1/b}$, where $det(M_i)$ is the determinant of the normalized dispersion matrix for the uncentered (centered) block submatrix of X, and b is the number of blocks. The normalization is with respect to the number of observations in each block.
diagonality or SS	The diagonality is $(\|M\|/P)^{1/k}$, where P is the product of the diagonal elements of M. The SS is the sum of squares of S when the OB criterion is used.

Blocks	A list of the blocks, labeled B1, B2, etc.
design	A data.frame. The design with blocks stacked in order.
rows	Numeric row numbers of the design rows corresponding to the withinData rows.
args	A list of the actual arguments used in this call.

Author(s)

Bob Wheeler <bwheelerg@gmail.com>

Please cite this program as follows:

Wheeler, R.E. (2004). optBlock. *AlgDesign*. The R project for statistical computing https://www.r-project.org/

References

Atkinson, A.C. and Donev, A.N. (1989). The construction of exact D-optimum experimental designs with application to blocking response surface designs. *Biometrika*. 76. 515-526.

Cook, R.D. and Nachtsheim, C.J. (1989). Computer-aided blocking of factorial and response-surface designs. *Technometrics.* 31-3. 339-346.

Goos, P. and Vandebroek, M. (2003). D-optimal split-plot designs with given numbers and sizes of whole plots. *Technometrics.* 45-3. 235-245.

Nguyen, Nam-Ky. (2001). Cutting experimental designs into blocks. *AusNZJSt*. 43-3. 367-374.

Trinca, L.A. and Gilmour, S.G. (2000). An algorithm for arranging response surface designs in small blocks. *Computational Statistics and Data Analysis*. 33. 25-43.

Trinca, L.A. and Gilmour, S.G. (2001). Multistratum response surface designs. *Technometrics*. 43-1. 25-33.

Examples

```
# Blocking the design for a quadratic polynomial in three variables into two
# seven trial blocks:

dat<-gen.factorial(3,3,varNames=c("A","B","C"))
desD<-optFederov(~quad(.),dat,nTrials=14,eval=TRUE) # Choose an optimum 14 trail design.
optBlock(~quad(.),desD$design,c(7,7))

# Letting optBlock() search the dat candidate list instead of first choosing a
# 14 trial design.
optBlock(~quad(.),dat,c(7,7))


# A block design for 7 treatments in 7 blocks of size 3. Note how withinData
# is recycled to fill out the blocksize requirements.

BIB<-optBlock(~.,withinData=factor(1:7),blocksizes=rep(3,7))

# This is a balanced incomplete block design as may be seen from:

crossprod(table(BIB$rows,c(rep(1:7, rep(3,7)))))
```

```
# A partially balanced incomplete block design with two associate classes:

tr<-factor(1:9)
PBIB<-optBlock(~.,withinData=tr,blocksizes=rep(3,9))

crossprod(table(PBIB$rows,c(rep(1:9, rep(3,9)))))


# Two fractions of a 2^(4-1).

dat<-gen.factorial(2,4)
od<-optBlock(~.,dat,c(8,8))

# The blocks are not themselves orthogonal even though the entire design is optimal.

bk<-data.matrix(od$Blocks$B1)
t(bk)%*%bk

# Better blocks may be obtained as follows, but note that they are not generally
# the fractions that would be obtained by confounding the third order interaction.

od<-optBlock(~.,dat,c(8,8),criterion="Dpc",nR=10)
bk<-data.matrix(od$Blocks$B1)
t(bk)%*%bk

# Blocking with whole plot factors. Note that the 27 rows of within are recycled
# to make the 54 trial blocked design.

within<-expand.grid(A=c(-1,0,1),B=c(-1,0,1),C=c(-1,0,1))
whole<-expand.grid(D=factor(1:3),E=factor(1:3))
od<-optBlock(~D+E*(quad(A,B,C)),withinData=within,blocksizes=rep(6,9),wholeBlockData=whole)

# Either withinData, or wholeBlockData may be an approximate theory optimial design
# produced by optFederov() for nTrials. The first column in the optFederov() output
# design, named "Rep..", is used to replicate the trials.

within<-optFederov(~quad(A,B,C),within,nT=54,approx=TRUE)
od<-optBlock(~D+E*(quad(A,B,C)),withinData=within$design,blocksizes=rep(6,9),wholeBlockData=whole)
```

---

optFederov *Optimal design*

Description

Calculates an exact or approximate algorithmic design for one of three criteria, using Federov's exchange algorithm.

Usage

```
optFederov(frml,data,nTrials,center=FALSE,approximate=FALSE,criterion="D",
evaluateI=FALSE,space=NULL,augment=FALSE,rows,nullify=0,
maxIteration=100,nRepeats=5,DFrac=1,CFrac=1,args=FALSE)
```

Arguments

frml	This may be omitted if data is the fully model expanded candidate list. If present it should be a formula starting with ~ which describes the model using variable names from data. It may be ~. if all variables from data are to be used linearly. In addition to the usual operators, quad(), cubic() and cubicS() may be used to expand variables from data into polynomial models.
data	The candidate list. A matrix or data.frame describing the variables. If a matrix is input and the columns are not named, they will be assigned names X1,X2, etc. If a data.frame is input without column names, they will be named Var1, Var2, etc. Although data may be input as global variables used in frml, it is preferable to input it here.
nTrials	If approximate=FALSE, it is the number of trials in the final design and if missing, it will be taken as the greater of length(rows) or 5 plus the number of terms in the model. If approximate=TRUE, nTrials will be used to round the optimal proportions so that the replications of the points add to nTrials.
approximate	When FALSE, an exact design in nTrails will be calculated. When TRUE the proportions for an approximate theory design will be calculated. If nTrials is set, any proportion less than 1/(2*maxIteration) will be discarded before the proportions are efficiently rounded, otherwise all non-zero proportions will be shown: these are the support points.
center	When TRUE, the numeric variables will be centered.
criterion	"D", "A", or "I"
evaluateI	TRUE to evaluate and report I in addition to other criteria. This parameter is included, because evaluating I requires extra effort.
space	If the criterion is "I" or evaluate I is true, the space over which the I criterion is to be evaluated may be input. It should be a matrix with the same column types and names as in data. If space is not input the evaluation will be done over the space described by data.
augment	If TRUE, the row numbers in rows will never be exchanged.
rows	Either a vector of row numbers (not row names) from data to be used as the starting design or a vector of row numbers for the design to be augmented. Note, replicate row numbers will be discarded and the length of rows cannot exceed the number of rows in data.
nullify	When non-zero, the initial design is obtained by nullification. If nullify=1, nTrials will be calculated (In this case nRepeats is set to 1). If nullify=2, number-of-terms trials will be calculated, and the remainder, up to nTrials, will be filled out at random.

maxIteration	maximum number of times points are exchanged, within each repeat, in seeking an optimum design.
nRepeats	Number of times the entire process is repeated. Has no effect when approximate=TRUE, or when nullify=1.
DFrac	Design fraction: the fraction of design used in search: 1 uses all of them, 0 uses only the one with the smallest variance.
CFrac	Candidate fraction: the fraction of candidate set searched : 1 uses all of them, 0 uses only the one with the largest variance.
args	If TRUE, the actual arguments to the function including the starting random number seed will be output.

Details

Let $E(y) = Zb$, where y is a vector of n observations, Z is an $n \times k$ matrix, and b is a vector of k parameters. The "exact" design problem is to find a matrix Z, with rows selected from a $N \times k$ matrix X, that is "best" in some sense. The matrix X can be a discretization of a continuous space or it can represent categories. In either case, the algorithmic design calculation is with respect to X, and not to some larger space containing the points.

Approximate designs weight the candidate points with a probability measure, which for practical purposes amounts to allowing unequal replication.

The Federov(1972) algorithm starts with n points chosen from X. They may be chosen randomly or by nullification, a procedure which iteratively adds points from the null space of X, until a non-singular n point design is found. The Federov algorithm exchanges points in the n point design Z with points in $X - Z$, i.e. points not in Z, in order to optimize a criterion, and quits when no profitable exchanges are possible, or the input parameter `maxIteration` is reached. The quality of the result depends on the starting design and the result may represent a local optimum. The procedure is repeated `nRepeats` times in order to come nearer to a global optimum. The parameters `DFrac` and `CFrac` control the portions of Z and $X - Z$ that are used.

The goal of algorithmic design is to maximize the information about the parameters. The information matrix is a matrix proportional to $M = Z'Z/n$, and various functions of M are chosen for optimization. The most popular of these is the `D` criterion, $|M|^{1/k}$, which is thus a scaling of the "generalized variance." Other criteria of interest involve the variance of predicted values, such as the `G` criterion, which is the minimax value of $d(x) = x'(Mi)x$, over X, where Mi is the inverse of M, and x' is a row of X; and the `I` criterion, which is the average value of $d(x)$ in the experimental region. These criteria are invariant under linear transformations of the parameter vector, which frees them from a dependency on units of scale. Other criteria are not invariant, such as the largest eigenvalue of M or the `A` criterion, which is $trace(Mi)/k$: it is of course proportional to the average variance of the parameter estimates. The criteria `D`, `A`, and `I` are supported by `optFederov()`, and `G`, which is intimately connected to `D`, is reported.

The theoretical optimum value of `G` is known for approximate theory designs, and so G_e, the `G` efficiency of `G` is available as a standard of design quality. It is especially useful, because G_e provides a lower bound on D_e, the `D` efficiency for approximate theory, to wit:

$$D_e \geq exp(1 - 1/G_e)$$

.

A vignette giving further details is availble. To access it, type

vignette("AlgDesign")

Value

D	The kth root of the generalized variance: $det(M)^{1/k}$, where $det(M)$ is the determinant of the normalized dispersion matrix M – i.e. $M = Z'Z/n$, where $Z = X[rows,]$
A	The average coefficient variance: $trace(Mi)/k$, where Mi is the inverse of M.
I	The average prediction variance over X, which can be shown to be $trace((X'X * Mi)/N)$, where N is the number of rows in X. This is calculated only when I is the criterion or when `evaluateI` is TRUE.
Ge	The minimax normalized variance over X, expressed as an efficiency with respect to the optimal approximate theory design. It is defined as $k/max(d)$, where $max(d)$ is the maximum normalized variance over X – i.e. the max of $x'(Mi)x$, over all rows x' of X.
Dea	A lower bound on `D` efficiency for approximate theory designs. It is equal to $exp(1 - 1/Ge)$.
design	The design.
rows	A numerical vector of the design row numbers.
args	A list of the actual arguments used in this call.

Note

Algorithmic design is often used with continuous and mixture variables for which R has minimal support, thus the functions quad(), cubic(), and cubicS() may be used in frml. The translation is done with expand.formula.

Mixture variables are variables such that the rows of `data` sum to a constant value, usually unity. Because of the mixture constraint, models containing mixture variables should not have a constant term. The linear and quadratic models for mixture variables A, B, and C are given by $-1 + (A + B + C)$ and $-1 + (A + B + C)^2$ respectively. See Gorman and Hinman [1962] for details.

The function gen.mixture() generates a list of candidate points whose rows sum to unity.

Author(s)

Bob Wheeler <bwheelerg@gmail.com>

Please cite this program as follows:

Wheeler, R.E. (2004). optFederov. *AlgDesign*. The R project for statistical computing https://www.r-project.org/

References

Atkinson, A.C. and Donev, A.N. (1992). *Optimum experimental designs*. Clarendon Press, Oxford.

Gorman, J.W. and Hinman, J.E. (1962). Simplex lattice designs for multicomponent systems. *Technometrics*. 4-4. 463-487.

Federov, V.V. (1972). *Theory of optimal experiments*. Academic Press, N.Y.

Examples

```
# EXAMPLE 1
# A quadratic polynomial in three variables. The resulting D will be about 0.46.
# This may be compared with a standard central composite design obtained from
# rows 1,3,5,7,9,11,13,15,17,19,21,23,25,27 of dat, which has a D value of 0.46.
# The central composite design seems to be the optimal design for all three criteria.

dat<-gen.factorial(levels=3,nVars=3,varNames=c("A","B","C"))

desD<-optFederov(~quad(A,B,C),dat,nTrials=14,eval=TRUE)
desA<-optFederov(~quad(.),dat,nTrials=14,eval=TRUE,crit="A")
desI<-optFederov(~quad(.),dat,nTrials=14,eval=TRUE,crit="I")

rows<-c(1,3,5,7,9,11,13,15,17,19,21,23,25,27)
desO<-optFederov(~quad(.),dat,nTrials=14,eval=TRUE,rows=rows)

# The I criterion may be seen to decrease as the space is expanded.

levels<-seq(-1,1,by=.1)
dat<-expand.grid(list(A=levels,B=levels,C=levels))

desL<-optFederov(~quad(.),dat,nTrials=14,eval=TRUE)

# This is not the case for A or D. For A and D, the support points are the points
# of the grid with the three levels above. Points not on this grid move
# the criteria in a non-optimal direction; hence, the enlarging space has no effect.

# EXAMPLES 2
# Standard designs are usually optimal designs. If nTrials is set to that for
# a standard design, and if nRepeats is large enough, the standard design will
# often be found For example, a half replicate of a 2^4 will be obtained by the
# following.

dat<-gen.factorial(levels=2,nVars=3,varNames=c("A","B","C"))
desH<-optFederov(~.,dat,8)

# A third replicate of a 3^3 will be obtained by the following:

dat<-gen.factorial(levels=3,nVars=3,factor=1:3)
desT<-optFederov(~.,dat,9)

# An orthogonal design similar to a 12 run Plackett-Burman design can be
# created by the following.

dat<-gen.factorial(levels=2,nVars=11,varNames=c("A","B","C","D","E","F","G","H","J","K","L"))
desPB<-optFederov(~.,dat,12,nRepeats=20)

# The above calculation is numerically difficult for the A and I criteria,
# and nRepeats=100 or more may be needed.

# It is instructive to examine a case in which the standard design is not found.
# The following is an attempt to create a Latin square design. It is not always successful.
```

```
lv<-factor(1:5)
dat<-expand.grid(A=lv,B=lv,C=lv)
desL<-optFederov(~.,dat,nTrials=25,nRep=100)

# It may be summarized as follows.

cs<-xtabs(~.,desL$design)
{xx<-matrix(0,5,5); for (i in 1:5) xx=xx+cs[1:5,1:5,i]*i;xx}



# EXAMPLE 3
# Mixture variables have a constant sum, usually 1. This causes a linear dependency
# among terms of polynomial models. In particular the constant term is dependent.
# Squared terms in a quadratic model are confounded with interaction terms, so that
# a quadratic model for three mixture variables is ~0+(A+B+C)^2. The following
# calculation generates a set of candidate varibles using gen.mixture() with
# four values on each axis, and then creates a 15 run design. The design is optimal.
# Indeed, the candidate set produced by gen.mixture(2,5) is optimal. Note:
# nullify=TRUE is used to ensure that this example will run withough error. The
# default value of 5 for nRepeats is sometimes not enought to find a starting
# design with a mixture problem.


dat<-gen.mixture(4,5)
desM<-optFederov(~(X1+X2+X3+X4+X5)^2-1,dat,15,nullify=TRUE)

# EXAMPLES 4
# Design augmenation can be obtained by setting augment=TRUE, and placing the row numbers
# of the design to be agmented in rows. Augmentation is often used to (1) add a new variable
# to an existing design or (2) to increase the complexity of the model. The following illustrates
# adding a variable to an existing design using desD above. It is assumed that all runs of the
# existing design have been made at the -1 level of the new variable:

dat<-gen.factorial(levels=3,nVars=3,varNames=c("A","B","C"))
desA<-optFederov(~quad(.),dat,nTrials=25,augment=TRUE,rows=desD$rows)

# The half fraction in desH, can be augmented to support an additional term:

dat<-gen.factorial(levels=2,nVars=4,varNames=c("A","B","C","D"))
desH<-optFederov(~.,dat,8)
desH2<-optFederov(~A+B+C+D+I(A*B),dat,10,aug=TRUE,rows=desH$rows)

# EXAMPLES 5
# Optimal approximate theory designs have non-zero probabilities only on support points.
# For the first example above the approximate theory design is as follows. It shows
# that all points in the cubic lattice are support points. The D for this
# design is 0.474 which may be compared with the D of 0.463 of the first example to
# indicate that that exact design had a D-efficiency of 97%. The lower bound Dea
# was 82%.

dat<-gen.factorial(levels=3,nVars=3,varNames=c("A","B","C"))
```

```
desDA<-optFederov(~quad(A,B,C),dat,eval=TRUE,approx=TRUE)

# The largest proportions will be rounded if nTrials is specified.

desDAN<-optFederov(~quad(A,B,C),dat,eval=TRUE,approx=TRUE,nTrials=15)
```

---

optMonteCarlo *Optimal design via Monte Carlo*

Description

Finds a design using the specified criterion via Federov's algorithm applied to a random subset of all possible candidate points.

Usage

```
optMonteCarlo(frml,data,nTrials,approximate=FALSE,criterion="D",evaluateI=FALSE,
space=NULL,mixtureSum=1,constraints=NULL,RandomStart=TRUE,nRepeats=5,nCand,
nCandNull,DFrac=1,CFrac=1,args=FALSE)
```

Arguments

frml	Required: A formula starting with ~ which will be used with model.matrix() to create a model matrix. If there are mixture variables, the constant term is suppressed.
data	Required: A data frame with 7 or 8 columns. See details below for specifics
nTrials	number trials in design – must be greater than the number of terms in the model, if missing will be set to the number of model terms in the model plus five.
approximate	When FALSE, an exact design in nTrails will be calculated. When TRUE the proportions for an approximate theory design will be calculated. If nTrials is set, any proportion less than 1/(2*maxIteration) will be discarded before the proportions are efficiently rounded, otherwise all non-zero proportions will be shown: these are the support points.
criterion	"D", "A", or "I"
evaluateI	TRUE if I is to be evaluated in addition to the other criteria – slower because of calculations for I
space	If the criterion is "I" or evaluate I is true, the space over which the I criterion is to be evaluated may be input. It should be a matrix with the same column types and names as in data. If space is not input the evaluation will be done over the space described by data.
constraints	A function taking a vector argument with length equal to the number of variables, and returning TRUE if the vector is inside the constrained region
mixtureSum	The mixture variables, if any, will sum to this value.

RandomStart	When TRUE, the starting design will be chosen at random, otherwise nullification will be used. Note: the nullifcation used here is different and much slower than that in optFederov().
nRepeats	number of times to retry the entire process
nCand	number of candidate points to generate, if missing, it will be 10 times the number of terms
nCandNull	Number of candidate points to use for nullification. If missing it will be set to nCand
DFrac	Fraction of design used in search: 1 uses all of them, 0 only the one with the smallest variance
CFrac	Fraction of candidate set searched : 1 uses all of them, 0 only the one with the largest variance
args	If TRUE, the actual arguments to the function including the starting random number seed will be output.

Details

The columns of the input data frame are as follows. The columns need not be named. It is probably best to avoid naming the variables with single letters, especially "I" – use paste(), as in the examples. For each variable nLevels are randomly generated between low and high, inclusive, and then rounded with round. For integer levels, round should be set to 0.

var: The names of the variables.

low: The lower limit of the range for each variable. Ignored for mixtures.

high: The upper limit of the range for each variable. Ignored for mixtures.

center: The centering value for each variable. Ignored for mixtures.

nLevels: The number of levels for each variable. Ignored for mixture variables.

round: The number of decimal digits for the levels. The levels are randomly and uniformly chosen between low and high, and this parameter controls the number of trailing digits. The max value for mixture variables in this vector is used to round all mixture variables.

factor: TRUE, FALSE depending on whether or not the variable is a factor. Note: other columns will be reset to conform to a nLevels factor.

mix: TRUE if the variable is a mixture variable. This column may be omitted if there are no mixture variables.

Candidate lists required by `optFederov()` increase with the number of variables, and can easily exceed storage capacity and can require excessive amounts of time to process. To overcome this problem, `optMonteCarlo()`, generates at random `nCand` points from a putative candidate list.

For non-mixture variables, `optMonteCarlo()` samples from the putative candidate list by choosing random levels inside the limits given by `low` and `high` in `data`. These are rounded to the number of levels given by `nLevels` in `data` and to the number of decimal digits given by `round` in `data`.

For mixture variables, `optMonteCarlo()` samples from the putative candidate list by choosing random levels between 0 and 1, rounded to the maximum in the `round` column of `data`, and such that the sum over all variables is equal to `mixtureSum`.

If a constraint function is supplied in `Constraints`, it is applied, and results which do not meet the constraint are discarded. The constraint function should be written to process uncentered variables.

The above procedures are repeated until `nCand` candidate points are found.

Nullification, successively adds points to a design until n points are found. This is the same procedure that is in `optFederov` except that each new point is selected from a new sampling of the putative candidate points. In general, this will produce better designs that those from a random start.

The entire process is repeated `nRepeats` times, and the best result is reported. The methodology compares favorably with an exhaustive search where the entire candidate list is searched by `optFederov()`.

The random numbers used in these calculations are controlled by the usual R random number mechanism.

A vignette giving further details is availble. To access it, type

vignette("AlgDesign")

Value

The output is the same list as from `optFederov`, but the criteria values are relative to the randomly chosen subsets of the putative candidate space. In general, they should not differ greatly from those obtained by an exhaustive search.

`D`	The kth root of the generalized variance: $det(M)^{1/k}$, where $det(M)$ is the determinant of the normalized dispersion matrix M – i.e. $M = Z'Z/n$, where $Z = X[rows,]$
`A`	The average coefficient variance: $trace(Mi)/k$, where Mi is the inverse of M.
`I`	The average prediction variance over X, which can be shown to be $trace((X'X * Mi)/N)$. This is calculated only when I is the criterion or when `evaluateI` is TRUE.
`Ge`	The minimax normalized variance over X, expressed as an efficiency with respect to the optimal approximate theory design. It is defined as $k/max(d)$, where $max(d)$ is the maximum normalized variance over X – i.e. the max of $x'(Mi)x$, over all rows x' of X.
`Dea`	A lower bound on `D` efficiency for approximate theory designs. It is equal to $exp(1 - 1/Ge)$.
`Design`	The design.
`args`	A list of the actual arguments used in this call.

Author(s)

Bob Wheeler <bwheelerg@gmail.com>

Please cite this program as follows:

Wheeler, R.E. (2004). optMonteCarlo. *AlgDesign*. The R project for statistical computing https://www.r-project.org/

Examples

```
# EXAMPLE 1
# The data.frame in data might look like the following:
data<-data.frame(var=paste("X",1:6,sep=""),low=c(1,1,1,0,0,0),
high=c(3,3,3,1,1,1),center=c(2,2,2,0,0,0),nLevels=3,
round=1,factor=0,mix=c(FALSE,FALSE,FALSE,TRUE,TRUE,TRUE))
data

# and the design:

optMonteCarlo(~(X1+X2+X3)^2+X4+X5+X6,data)

# Example 2
# Standard designs will often be produced, just as
# they will with optFederov(). For example,
# a half fraction of a 2^4:
data<-data.frame(paste("X",1:4,sep=""),-1,1,0,2,0,0)
data
optMonteCarlo(~.,data,nTrials=8)

# Example 3
# optMonteCarlo() can treat much larger problems than can
# optFederov().  For example, optFederov()
# requires a candidate list of 3^20 points for
# a 20 variable, 3 level candidate list -- about
# 25 gigabytes. If the model is quadratic, this must
# be multiplied by about 12. There are other storage
# requirements internal to optFederov() which easily
# double this value. optMonteCarlo() since it only samples
# from the putative candidate list, has no difficulty
# with a problem of this size. The criterion values
# appearing in the output of optMonteCarlo() are based on
# these samples, but their values seem to be reasonable
# correct, as the following shows: (These are commented
# out for those who have a slow machine.)

dat<-gen.factorial(levels=3,nVar=8)
#desF<-optFederov(~quad(.),dat,eval=TRUE)
#desF[1:5]

data<-data.frame(paste("X",1:8,sep=""),-1,1,0,3,0,0)
#desH<-optMonteCarlo(~quad(.),data,Rand=FALSE,eval=TRUE)
#desH[1:5]

# The following is a 20 variable quadratic. Uncomment
# and wait a while, even if you have a fast machine.
# Note: nRepeats has been changed from its default.
# Note: criterion values for exact designs are often
# far from approximate theory optima; hence, Ge and De
# will be small.

data<-data.frame(paste("X",1:20,sep=""),-1,1,0,3,0,0)
```

```
#desBig<-optMonteCarlo(~quad(.),data,nRepeats=1)

# The following will produce improved criterion values

#desNBig<-optMonteCarlo(~quad(.),data,Rand=FALSE,nRepeats=1)

# EXAMPLE 4
# Practically infeasible combinations of variable are
# common. Designs may be produced which avoid such
# combinations by using a constraint function. Suppose,
# for example that one corner of a cubic box is not
# feasible, then the following will produce a design
# that makes no use of this corner.

Constraints<-function(x){!(x[1]>0.75 && x[2]>0.75)}
data<-data.frame(paste("X",1:4,sep=""),-1,1,0,3,0,0)
desC<-optMonteCarlo(~.,data,con=Constraints)

# The above just removes a corner. Increasing the
# number of levels will remove points along the
# boundary.

data<-data.frame(paste("X",1:4,sep=""),-1,1,0,11,3,0)
desC2<-optMonteCarlo(~.,data,con=Constraints)
```

TGTable3 *Trinca Gilmour Table 3*

Description

A blocked experimental design from Trinca and Gilmour (2001).

A data.frame giving four three level variable in five blocks. There are 45 trials in all.

Usage

```
data(TGTable3)
```

Source

Trinca, L.A. and Gilmour, S.G. (2001). Multistratum response surface designs. *Technometrics*. 43-1. 25-33.

TGTable5 *Trinca Gilmour Table 5*

Description

A blocked experimental design from Trinca and Gilmour (2001).

A matrix giving one whole plot and four within plot variables, in 21 blocks of 2.

Usage

```
data(TGTable5)
```

Source

Trinca, L.A. and Gilmour, S.G. (2001). Multistratum response surface designs. *Technometrics*. 43-1. 25-33.

Index