Package 'DoE.wrapper'

September 29, 2014

Title Wrapper package for design of experiments functionality

Version 0.8-10

Depends R(>= 2.13.0), FrF2(>= 1.6-5), DoE.base(>= 0.23-4), rsm

Imports lhs, DiceDesign, AlgDesign(>= 1.1)

Date 2014-09-29

Description This package creates various kinds of designs for (industrial) experiments. It uses, and sometimes enhances,design generation routines from other packages. So far, response surface designs from package rsm, latin hypercube samples from packages lhs and DiceDesign, and D-optimal designs from package AlgDesign have been implemented.

License GPL (>= 2)

LazyLoad yes

LazyData yes

Encoding latin1

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

Author Ulrike Groemping [aut, cre],Lenth Russ [ctb]

Maintainer Ulrike Groemping <groemping@beuth-hochschule.de>

NeedsCompilation no

Repository CRAN

Date/Publication 2014-09-29 23:16:57

R topics documented:

`bbd.design` *Function for generating Box-Behnken designs*

Description

Function for generating Box-Behnken designs, making use of package rsm

Usage

```
bbd.design(nfactors, ncenter=4, factor.names = NULL, default.levels=c(-1,1),
          block.name=NULL, randomize=TRUE, seed=NULL, ...)
```

Arguments

`nfactors`	number of factors
`ncenter`	integer number of center points for each block
`factor.names`	list of scale end values for each factor (the middle value is calculated); names are used as variable names; **the names must not be x1, x2, ..., as these are used for the variables in coded units**; if the list is not named, the variable names are A, B and so forth; in the coded units used in attribute desnum, `-1` corresponds to the smaller, `+1` to the larger value.
`default.levels`	default levels (vector of length 2) for all factors for which no specific levels are given; must consist of two numeric values for the scale ends, the default middle level is calculated
`block.name`	name of block factor that distinguishes between the blocks; blocks are usable for nfactors=4 and nfactors=5 only, `block.name` is ignored otherwise.
`randomize`	logical that indicates whether or not randomization should occur
`seed`	seed for random number generation in randomization
`...`	reserved for future usage

Details

Function bbd.design creates a Box-Behnken design, which is a design for quantitative factors with all factors on three levels. Box-Behnken designs should not be used if the combinations of the extreme levels of the factors are of interest (cf. also Myers, Montgomery and Anderson-Cook 2009). There are designs for 3 to 7 factors, and the unreplicated versions of these have 14 (3 factors), 24 (4 factors), 40 (5 factors), 48 (6 factors), and 56 (7 factors) runs plus the specified number of center points ncenter.

Function bbd.design is an interface to function bbd from package **rsm** that makes this design accessible using similar syntax as used in packages **DoE.base** and **FrF2** and creates an output object that is also treatable by the convenience functions available in package **DoE.base**.

Currently, creation of replications and repeated measurements - as would be usual for other design functions - is not implemented. This is planned for the future, but does not have high priority.

Value

The function returns a data frame of S3 class design with attributes attached. The data frame itself is in the original data scale. The matrix desnum attached as attribute desnum contains the coded data, the attribute run.order contains the standard order and the actual run order of the design (contrary to package rsm, the row names of the design refer to the actual rather than the standard run order).

The attribute design.info is a list of design properties. The element type of that list is the character string bbd. Besides the elements present in all class design objects, there are the elements quantitative (vector with nfactor TRUE entries), and a codings element usable in the coding functions available in the **rsm** package, e.g. coded.data.

Note

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

Author(s)

Ulrike Groemping

References

Box, G.E.P. and Behnken, D.W. (1960). Some new three-level designs for the study of quantitative variables. *Technometrics* **2**, 455-475.

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

Box, G.E.P. and Wilson, K.B. (1951). On the Experimental Attainment of Optimum Conditions. *J. Royal Statistical Society*, **B13**, 1-45.

NIST/SEMATECH e-Handbook of Statistical Methods, http://www.itl.nist.gov/div898/handbook/pri/section3/pri3361.htm, accessed August 20th, 2009.

Myers, R.H., Montgomery, D.C. and Anderson-Cook, C.M. (2009). *Response Surface Methodology. Process and Product Optimization Using Designed Experiments*. Wiley, New York.

See Also

See also FrF2, ccd.design, lhs-package, rsm

Examples

```
plan1 <- bbd.design(5)  ## default for 5 factors is unblocked design, contrary to package rsm
plan1
## blocked design for 4 factors, using default levels
plan2 <- bbd.design(4,block.name="block",default.levels=c(10,30))
plan2
desnum(plan2)
## design with factor.names and modified ncenter
bbd.design(3,ncenter=6,
  factor.names=list("one"=c(25,35),"two"=c(-5,20), "three"=c(20,60)))
## design with character factor.names and default levels
bbd.design(3,factor.names=c("one","two", "three"), default.levels=c(10,20))
```

---

ccd.augment *Function for augmenting an existing cube with a star portion, using package rsm*

Description

Function for augmenting an existing fractional factorial with a star portion in case of a late decision for a sequential procedure.

Usage

```
ccd.augment(cube, ncenter = 4, columns="all", block.name="Block.ccd",
        alpha = "orthogonal", randomize=TRUE, seed=NULL, ...)
```

Arguments

cube	design generated by function FrF2. The design must not be a split-plot design, nor a parameter design in long version.
ncenter	integer number of center points, or vector with two numbers, the first for the cube and the second for the star portion of the design. If only one number is given, this is either used for each block (if the cube block does not have center points yet) or for the star portion of the design only.
block.name	name of block factor that distinguishes (at least) between blocks; even for unblocked cubes, the ccd design has a cube and a star point block
alpha	"orthogonal", "rotatable", or a number that indicates the position of the star points; the number 1 would create a face-centered design.
randomize	logical that indicates whether or not randomization should occur
seed	NULL or a vector of two integer seeds for random number generation in randomization

...	reserved for future usage
columns	not yet implemented; it is intended to later allow to add star points for only some factors of a design (after eliminating the others as unimportant in a sequential process), and columns will be used to indicate those

Details

The statistical background of central composite designs is briefly described under CentralCompositeDesigns.

Function ccd.augment augments an existing 2-level fractional factorial that should already have been run with center points and should have resolution V.

In exceptional situations, it may be useful to base a ccd on a resolution IV design that allows estimation of all 2-factor interactions of interest. Thus, it can be interesting to apply function ccd.augment to a cube based on the estimable functionality of function FrF2 in cases where a resolution V cube is not feasible. Of course, this does not allow to estimate the aliased 2-factor interactions and therefore generates a warning.

Value

The function returns a data frame of S3 class design with attributes attached. The data frame itself is in the original data scale. The data frame desnum attached as attribute desnum is the original data frame returned by package rsm. The attribute design.info is a list of various design properties. The element type of that list is the character string ccd. Besides the elements present in all class design objects, there are the elements quantitative (vector with nfactor TRUE entries), and a codings element usable in the coding functions available in the **rsm** package, e.g. coded.data.

Note that the row names and the standard order column in the run.order attribute of ccd designs based on estimability requirements (cf. also the details section) are not in conventional order and should not be used as the basis for any calculations. The same is true for blocked designs, if the blocking routine blockpick.big was used.

Note

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

Author(s)

Ulrike Groemping

References

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

Box, G.E.P. and Wilson, K.B. (1951). On the Experimental Attainment of Optimum Conditions. *J. Royal Statistical Society*, **B13**, 1-45.

NIST/SEMATECH e-Handbook of Statistical Methods, http://www.itl.nist.gov/div898/handbook/pri/section3/pri3361.htm, accessed August 20th, 2009.

Myers, R.H., Montgomery, D.C. and Anderson-Cook, C.M. (2009). *Response Surface Methodology. Process and Product Optimization Using Designed Experiments*. Wiley, New York.

See Also

See also ccd.design, FrF2, lhs-package, rsm

Examples

```
## purely technical examples for the sequential design creation process
  ## start with a fractional factorial with center points
  plan <- FrF2(16,5,default.levels=c(10,30),ncenter=6)
  ## collect data and add them to the design
  y <- rexp(22)
  plan <- add.response(plan,y)
  ## assuming that an analysis has created the suspicion that a second order
  ## model should be fitted (not to be expected for the above random numbers):
  plan.augmented <- ccd.augment(plan, ncenter=4)
  ## add new responses to the design
  y <- c(y, rexp(14))  ## append responses for the 14=5*2 + 4 star points
  r.plan.augmented <- add.response(plan.augmented, y, replace=TRUE)

## for info: how to analyse results from such a desgin
  lm.result <- lm(y~Block.ccd+(.-Block.ccd)^2+I(A^2)+I(B^2)+I(C^2)+I(D^2)+I(E^2),
                   r.plan.augmented)
  summary(lm.result)
  ## analysis with function rsm
  rsm.result <- rsm(y~Block.ccd+SO(A,B,C,D,E), r.plan.augmented)
  summary(rsm.result)  ## provides more information than lm.result
  loftest(rsm.result)  ## separate lack of fit test
  ## graphical analysis
  ## (NOTE: purely for demo purposes, the model is meaningless here)
  ## individual contour plot
  contour(rsm.result,B~A)
  ## several contour plots
  par(mfrow=c(1,2))
  contour(rsm.result,list(B~A, C~E))
  ## many contourplots, all pairs of some factors
  par(mfrow=c(2,3))
  contour(rsm.result,~A+B+C+D)
```

---

ccd.design *Function for accessing central composite designs from package rsm*

Description

Function for accessing central composite designs from package rsm, with automatic creation of an appropriate cube portion

Usage

```
ccd.design(nfactors=NULL, factor.names=NULL, default.levels=c(-1,1), ncube=NULL,
        resolution=if (identical(blocks,1) & is.null(ncube)) 5 else NULL,
```

```
          generators=NULL, ncenter = 4, alpha = "orthogonal",
          replications=1,
          block.name="Block.ccd", blocks=1,
          randomize=TRUE, seed=NULL, ...)
```

Arguments

nfactors	number of factors
factor.names	list of cube corner values for each factor; names are used as variable names; **the names must not be x1, x2, ..., as these are used for the variables in coded units**; if the list is not named, the variable names are X1, X2 and so forth; in coded units, -1 corresponds to the smaller, +1 to the larger value.
default.levels	default levels (vector of length 2) for all factors for which no specific levels are given
ncube	integer number of cube points (without center points for the cube)
resolution	arabic numeral for the requested resolution of the cube portion of the design; cubes for ccd designs should usually be at least of resolution V. the default value for resolution is therefore 5, unless generators or blocks are specified, in which case the default is NULL
generators	generators in the form allowed in function FrF2
ncenter	integer number of center points for each cube or star point block, or vector with two numbers, the first for the cube and the second for the star portion of the design
alpha	"orthogonal", "rotatable", or a number that indicates the position of the star points; the number 1 would create a face-centered design.
replications	the number of replications of the design; currently, only proper replications can be generated; these are randomized in blocks within the center point and star blocks. The same number of replications is used for both the cube and the star blocks.
block.name	name of block factor that distinguishes between blocks; even for unblocked cubes, the ccd design has at least one cube and one star point block
blocks	the same as in function FrF2; is EITHER the number of blocks into which the experiment is subdivided OR a character vector of names of independent factors that are used as block constructors OR a vector or list of generators similar to generators. In the latter case, the differences to generators are • that numbers/letters refer to the factors of the experiment and not to column numbers of the Yates matrix • that numbers/letters can refer to *all* nfactors factors rather than the log2(nruns) base factors only, • that one single number is always interpreted as the number of blocks rather than a column reference,

- that individual numbers are allowed in a list (i.e. individual factors specified in the experiment can be used as block factors) and
- that no negative signs are allowed.

If `blocks` is a single number, it must be a power of 2. A blocked design can have at most `nruns-blocks` treatment factors, but should usually have fewer than that.

If the experiment is randomized, randomization happens within blocks.

For the statistical and algorithmic background of blocked designs, see `block`.

`randomize`	logical that indicates whether or not randomization should occur
`seed`	NULL or a vector of two integer seeds for random number generation in randomization
`...`	reserved for future usage

Details

The statistical background of central composite designs is briefly described under `CentralCompositeDesigns`.

Function `ccd.design` creates a central composite design from scratch. It proceeds by generating a cube design with function `FrF2` and then augmenting this cube design using functions `add.center` from package **FrF2** for adding center points to the cube and subsequently function `ccd` from package **rsm** for generating the star portion of the design.

There are two main purposes for this function: one is to provide central composite designs within the same syntax philosophy used in packages `DoE.base-package` and `FrF2`. The other is to automatically identify good (=resolution V) cube portions, which can be achieved by using the resolution parameter.

In comparison to direct usage of package **ccd**, the functions make the syntax closer to that of the other packages in the DoE.wrapper suite and allow automatic selection of fractional factorials as cubes.

Function `ccd.design` does not allow direct use of the `estimable` functionality that is available in function `FrF2`. Nevertheless, ccd designs with a cube based on the `estimable` functionality can be generated by first using function `FrF2` and subsequently applying function `ccd.augment`. It may for example be interesting to use designs based on estimability requirements for 2-factor interactions in cases where a resolution V cube for the ccd is not feasible - of course, this does not allow to estimate the full second order model and therefore generates a warning.

Value

The function returns a data frame of S3 class `design` with attributes attached. The data frame itself is in the original data scale. The data frame `desnum` attached as attribute `desnum` is the coded design. The attribute `design.info` is a list of various design properties. The element `type` of that list is the character string `ccd`. Besides the elements present in all class `design` objects, there are the elements quantitative (vector with `nfactor` TRUE entries), and a `codings` element usable in the coding functions available in the **rsm** package, e.g. `coded.data`.

Note that the row names and the standard order column in the `run.order` attribute of ccd designs are not in conventional order, if the blocking routine `blockpick.big` was used. In such situations, these should not be used as the basis for any calculations.

Note

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

Author(s)

Ulrike Groemping

References

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

Box, G.E.P. and Wilson, K.B. (1951). On the Experimental Attainment of Optimum Conditions. *J. Royal Statistical Society*, **B13**, 1-45.

NIST/SEMATECH e-Handbook of Statistical Methods, http://www.itl.nist.gov/div898/handbook/pri/section3/pri3361.htm, accessed August 20th, 2009.

Myers, R.H., Montgomery, D.C. and Anderson-Cook, C.M. (2009). *Response Surface Methodology. Process and Product Optimization Using Designed Experiments*. Wiley, New York.

See Also

See also ccd.augment, add.center, FrF2, lhs-package, rsm

Examples

```
ccd.design(5) ## per default uses the resolution V design in 16 runs for the cube
ccd.design(5, ncube=32) ## uses the full factorial for the cube
ccd.design(5, ncenter=6, default.levels=c(-10,10))
## blocked design (requires ncube to be specified)
ccd.design(5, ncube=32, blocks=4)
## there is only one star point block

## for usage of other options, look at the FrF2 documentation
```

CentralCompositeDesigns

Statistical background of central composite designs

Description

Brief description of the statistical background of central composite designs

Details

Central composite designs (ccd's) were invented by Box and Wilson (1951) for response surface experimentation with quantitative factors. They are used for estimation of second order response surface models, i.e. models that allow to estimate linear, quadratic and interaction effects for all factors.

Central composite designs consist of a cube and star points (also called axial points). Both the cube and the star portion of the design should have some center points. The cube is a (fractional) factorial design and should be at least of resolution V. The line between the center points and the star points intersects the faces of the cube in their middle (see the link to the NIST/Sematech e-Handbook for a visualization). There are two star points per factor, i.e. the number of runs for (each block of) the star portion of the design is twice the number of factors plus the number of center points in the star portion.

The tuning parameter `alpha` determines whether the star points lie on the faces of the cube (`alpha=1`, face-centered), inside the cube (`alpha<1`, inscribed) or outside the cube (`alpha>1`, circumscribed). The latter case is the usual one. The value of `alpha` can be chosen such that the design is rotatable (may be useful if the scales of the factors are comparable) or such that the design is orthogonally blocked (i.e. the block effects do not affect the effect estimates of interest). The default is to generate orthogonally blocked designs.

Central composite designs are particularly useful in sequential experimentation, where a (fractional) factorial with center points is followed up by a star portion of the design. While the cube can already estimate the linear and interaction effects, the center points can only estimate the sum of all quadratic effects. If this indicates that quadratic effects are important, a star portion can be added in order to investigate the model more deeply.

Note

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

Author(s)

Ulrike Groemping

References

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

Box, G.E.P. and Wilson, K.B. (1951). On the Experimental Attainment of Optimum Conditions. *J. Royal Statistical Society*, **B13**, 1-45.

NIST/SEMATECH e-Handbook of Statistical Methods, http://www.itl.nist.gov/div898/handbook/pri/section3/pri3361.htm, accessed August 20th, 2009.

Myers, R.H., Montgomery, D.C. and Anderson-Cook, C.M. (2009). Response Surface Methodology. Process and Product Optimization Using Designed Experiments. Wiley, New York.

See Also

See Also `ccd`, `ccd.design`, `ccd.augment`

DoE.wrapper-page *Wrapper package for design of experiments functionality*

Description

This package creates various kinds of designs for (industrial) experiments. It uses, and sometimes enhances, design generation routines from other packages. So far, response surface designs from package rsm, designs for computer experiments (latin hypercube samples etc.) from packages lhs and DiceDesign and D-optimal designs from package AlgDesign have been implemented.

Details

Currently, the package provides classical response surface designs from package **rsm**: Box-Behnken designs by Box and Behnken (function `bbd.design` and central composite designs by Box and Wilson (`ccd.design`) are implemented. For the latter, there is also a function for augmenting 2-level fractional factorials into central composite designs (`ccd.design`).

Furthermore, latin hypercube samples and other designs for computer experiments from packages **lhs** and **DiceDesign** are provided.

Furthermore, D-optimal designs have been implemented, using package **AlgDesign**. This implementation is currently in beta shape.

All designs created by this package are class `design` objects, which are data frames with attributes.

Apart from providing designs, the package also provides functions for comparing the quality of several designs with quantitative variables (function `compare`).

Note

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

Author(s)

Ulrike Groemping

References

Box, G.E.P. and Behnken, D.W. (1960). Some new three-level designs for the study of quantitative variables. *Technometrics* **2**, 455-475.

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

Box, G.E.P. and Wilson, K.B. (1951). On the Experimental Attainment of Optimum Conditions. *J. Royal Statistical Society*, **B13**, 1-45.

NIST/SEMATECH e-Handbook of Statistical Methods, http://www.itl.nist.gov/div898/handbook/pri/section3/pri3361.htm, accessed August 20th, 2009.

Myers, R.H., Montgomery, D.C. and Anderson-Cook, C.M. (2009). *Response Surface Methodology. Process and Product Optimization Using Designed Experiments*. Wiley, New York.

See Also

See also ccd.design, ccd.augment, bbd.design, lhs.design, Dopt.design, Dopt.augment, lhs-package, rsm, optFederov, optBlock, compare, FrF2

Dopt.augment *Function for augmenting a design with D-optimal additional points using optFederov from package AlgDesign*

Description

Function for comfortably augmenting a design with D-optimal additional points; this functionality is still somewhat experimental.

Usage

```
Dopt.augment(design, m=1, formula=NULL, candidates=NULL, constraint=NULL,
    center=FALSE, nRepeats=5,
    seed=NULL, randomize=TRUE, ...)
```

Arguments

design	an experimental design of class design, which may not be a blocked, split-plot, neither crossed or parameter design; it also must not be replicated with repeat.only replications.
m	integer number of additional points to add to design design
formula	a model formula (starting with a tilde), for the estimation of which a D-optimal design is sought; it can contain all column names from data or elements or element names from factor.names, respectively; usage of the "."-notation for "all variables" from data or factor.names is possible. The default formula (if the value NULL is not changed) is the formula associated with the design (by function formula.design). For quantitative factors, functions quad() and cubic describe the full quadratic or full cubic model in the listed variables (cf. examples and the expand.formula-function from package **AlgDesign**).
candidates	data frame of candidate points; if not specified, candidates are constructed as a full factorial from the factor.names element of the design.info attribute of design
constraint	a condition (character string!) used for reducing the candidate set to admissible points only. constraint is evaluated on the specified data set or after automatic creation of a full factorial candidate data set. The variable names from data or factor.names can be used by the constraint. Per default (i.e. if the constraint is NULL), the constraint attribute from design

is used.
If a previously-applied constraint is to be removed, specify `constraint = ""`.
It is not possible to apply a constraint that is already violated by the design that is to be augmented.

`center` requests that optimization is run for the centered model; the design is nevertheless output in non-centered coordinates

`nRepeats` number of independent repeats of the design optimization process; increasing this number may improve the chance of finding a global optimum, but will also increase search time

`seed` seed for generation and randomization of the design (integer number);
here, the seed is needed even if the design is not randomized, because the generation process for the optimum design involves random numbers, even if the order of the final design is not randomized;
if a reproducible design is needed, it is therefore recommended to specify a seed

`randomize` logical deciding whether or not the design should be randomized; if it is `TRUE`, the design (or the additional portion of the design) returned by the workhorse function `optFederov` is brought into random order after generation. Note that the generation process itself contains a random element per default; if exact repeatability for the returned design is desired, it is necessary to specify a seed (option `seed`) if in the case `randomize=FALSE`.

`...` additional arguments to function `optFederov` from package **AlgDesign**;
interesting arguments: `maxIteration`, `nullify` (calculate good starting design, especially set to 1, in which case `nRepeats` is set to 1;
arguments `criterion` and `augment` are not available, neither are `evaluateI`, `space`, or `rows`, and `args` does not have an effect.

Details

Function `Dopt.augment` augments an existing design by `m` D-optimal additional points (unblocked designs, no split-plot, no parameter or crossed design, no repeat.only replications), i.e. by points that make the design particularly efficient for the intended model.

Option `center`, which is available for both blocked and unblocked designs as part of the ... argument, requests optimization for the centered model; the design that is created is nevertheless an uncentered design.

NULL entries in the arguments are filled with automatic values that are determined from `design`.

Value

The function returns a data frame of S3 class `design` with attributes attached. The data frame contains the experimental settings. The matrix `desnum` attached as attribute `desnum` contains the model matrix of the design, using the formula as specified in the call.
Function `Dopt.augment` preserves additional variables (e.g. responses) that have been added to the design `design` before augmenting. Note, however, that the response data are NOT used in deciding about which points to augment the design with.

The attribute `run.order` provides the run number in standard order (as returned from function `optFederov` in package **AlgDesign**) as well as the randomized actual run order. The third column

is always identical to the first. Note that the first n runs (the ones that are already present before augmentation) have run numbers in standard order from 1 to n (i.e. their original run numbers in standard order, if they were also generated by Dopt.design are lost).

The attribute design.info is a list of various design properties, with type resolving to "Dopt.augment". In addition to the standard list elements (cf. design), the element quantitative is a vector of nfactor logical values or NAs, and the optional digits elements indicates the number of digits to which the data were rounded. The list contains further entries regarding the optimality that has been achieved (D, Dea and A).

Note that the original design is contained in the first rows of the new data set. The original design also contains columns that are not directly part of the design, e.g. comment columns.
Note that replications is always set to 1, even if the original design was replicated, and repeat.only is always FALSE. These elements are only present to fulfill the formal requirements for class design.)

Note

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

Author(s)

Ulrike Groemping

References

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

Federov, V.V. (1972). *Theory of optimal experiments*. Academic Press, New York.

Wheeler, R.E. (2004). *Comments on algorithmic design*. Vignette accompanying package **AlgDesign**. ../../AlgDesign/doc/AlgDesign.pdf.

See Also

See also optFederov, fac.design, quad, cubic, Dopt.design

Examples

```
 ## a full quadratic model with constraint in three quantitative factors
plan <- Dopt.design(36,factor.names=list(eins=c(100,250),zwei=c(10,30),drei=c(-25,25)),
                        nlevels=c(4,3,6),
                        formula=~quad(.),
                        constraint="!(eins>=200 & zwei==30 & drei==25)")
summary(plan)
y <- rnorm(36)
r.plan <- add.response(plan, y)
plan2 <- Dopt.augment(r.plan, m=10)
summary(plan2)
## add the new response values after conducting additional experiments
y <- c(y, rnorm(10))
r.plan2 <- add.response(plan2,y, replace=TRUE)
```

```
summary(r.plan2, brief=FALSE)
```

`Dopt.design` *Function for creating D-optimal designs with or without blocking from package AlgDesign*

Description

Function for comfortably creating a D-optimal design with or without blocking based on functions optFederov or optBlock from package AlgDesign; this functionality is still somewhat experimental.

Usage

```
Dopt.design(nruns, data=NULL, formula=~., factor.names=NULL, nlevels=NULL,
   digits=NULL, constraint=NULL, center=FALSE, nRepeats=5, seed=NULL, randomize=TRUE,
    blocks=1, block.name="Blocks", wholeBlockData=NULL, ...)
```

Arguments

`nruns`	number of runs in the requested design
`data`	data frame or matrix of candidate design points; if `data` is specified, `factor.names` and `levels` are ignored
`formula`	a model formula (starting with a tilde), for the estimation of which a D-optimal design is sought; it can contain all column names from `data` or elements or element names from `factor.names`, respectively; usage of the "."-notation for "all variables" from `data` or `factor.names` is possible. The default formula linearly includes all main effects for columns of `data` or factors from `factor.names` respectively, by using the "."-notation. Note that the variables from `wholeBlockData` must be explicitly included into the formula and are not covered by the "."-notation for "all variables". (Thus, the default formula does not work, if `wholeBlockData` is used.) For quantitative factors, functions `quad()` and `cubic` describe the full quadratic or full cubic model in the listed variables (cf. examples and the `expand.formula`-function from package **AlgDesign**).
`factor.names`	is used for creating a candidate set (for the within Block factors) with the help of function `fac.design`, if `data` is not specified. It is a list of vectors which contain - individual levels - or (in case of numerical values combined with nlevels) lower and upper scale end values for each factor. The element names are used as variable names; if the list is not named, the variable names are A, B and so forth (from function

fac.design).
factor.names can also be a character vector. In this case, nlevels must be specified, and levels are automatically assigned as integers starting with 1.

nlevels — can be omitted if the list factor.names explicitly lists all factor levels (which of course defines the number of levels).
For numeric factors for which factor.names only specifies the two scale ends, these are filled with equally-spaced intermediate points, using the nlevels entry as the length.out argument to function seq.
If factor.names is a character vector of factor names only, nlevels is required, and default levels are created.

digits — is used for creating a candidate set if data is not specified.
It specifies the digits to which numeric design columns are rounded in case of automatic creation of intermediate values. It can consist of one single value (the same for all such factors) or a numeric vector of the same length as factor.names with integer entries.

constraint — a condition (character string!) used for reducing the candidate set to admissible points only. constraint is evaluated on the specified data set or after automatic creation of a full factorial candidate data set.
The variable names from data or factor.names can be used by the constraint.
The variable names from wholePlotData can NOT be used.
See Syntax and Logic for an explanation of the syntax of general and especially logical R expressions.

center — requests that optimization is run for the centered model; the design is nevertheless output in non-centered coordinates

nRepeats — number of independent repeats of the design optimization process; increasing this number may improve the chance of finding a global optimum, but will also increase search time

seed — seed for generation and randomization of the design (integer number);
here, the seed is needed even if the design is not randomized, because the generation process for the optimum design involves random numbers, even if the order of the final design is not randomized;
if a reproducible design is needed, it is therefore recommended to specify a seed

randomize — logical deciding whether or not the design should be randomized; if it is TRUE, the design (or the additional portion of the design) returned by the workhorse function optFederov is brought into random order after generation. Note that the generation process itself contains a random element per default; if exact repeatability for the returned design is desired, it is necessary to specify a seed (option seed) if in the case randomize=FALSE.

blocks — a single integer giving the number of blocks (default 1, if no blocking is needed)
OR
a vector of block sizes which enable blocks of different sizes;
for a scalar value, nruns must be divisible into blocks equally-sized blocks; for a vector value, the block sizes must add up to nruns.
If blocking is requested, the following two options are potentially important.

block.name — character string: name of the blocking variable (used only if blocks are requested)

`wholeBlockData` optional matrix or data frame that specifies the whole block characteristics; can only be used if blocks are requested; if used, it must have as many rows as there are block sizes.
If this is specified, the resulting design is a split-plot design with the whole-plot factors specified in wholeBlockData, the split-plot factors specified in data. Note that usage of this option makes it necessary to explicitly specify a formula.

Since wholeBlockData must be completely specified by the user, optimization is for the split-plot portion of the design only. The rationale is (assumably) that the characteristics of the available blocks are known. If this is not the case, users may want to try out various possible whole block setups, or to proceed sequentially by first optimizing a whole block design for a model with the whole block factors only and subsequently using this model for adding split-plot factors.

`...` additional arguments to functions `optFederov` or `optBlock` (if blocking is requested) from package **AlgDesign**;
interesting arguments for `optFederov`: `maxIteration`, `nullify` (calculate good starting design, especially set to 1, in which case `nRepeats` is set to 1);
arguments `criterion` and `augment` are not available, neither are `evaluateI`, `space`, or `rows`, and `args` does not have an effect.

Details

Function `Dopt.design` creates a D-optimal design, optionally with blocking, and even as a split-plot design. If no blocks are required, calculations are carried out through function `optFederov` from package **AlgDesign**. In case of blocked designs, function `optBlock` from package **AlgDesign** is behind the calculations. By specifying `wholeBlockData`, a blocked design becomes a split-plot design. The model formula can refer to both the within block data (only those are referred to by the "." notation) and the whole block data and interactions between both.
In comparison to direct usage of package **AlgDesign**, the function adds the possibility of automatically creating the candidate points on the fly, with or without constraints. Furthermore, it embeds the D-optimal designs into the class `design`. On the other hand, it sacrifices some of **AlgDesign**s flexibility; of course, users can still use **AlgDesign** directly.

The D-optimal designs are particularly useful, if the classical regular designs are too demanding in run size requirements, or if constraints preclude automatic generation of orthogonal designs. Note, however, that the best design in few runs can still be very bad in absolute terms!

When specifying the design without the `data` option, a full factorial in the requested factors is the default candidate set of design points. For some situations - especially with many factors - it may be better to start from a restricted candidate set. Such a candidate set can be produced with another R function, e.g. `oa.design` or `FrF2`, or can be manually created.

If there are doubts, whether the process has delivered a design close to the absolute optimum, `nRepeats` can be increased.

For unblocked designs, it is additionally possible to increase `maxIteration`. Also, improving the starting value by `nullify=1` or `nullify=2` may lead to an improved design. These options are handed through to function `optFederov` from package **AlgDesign** and are documented there.

Value

The function returns a data frame of S3 class design with attributes attached. The data frame contains the experimental settings. The matrix desnum attached as attribute desnum contains the model matrix of the design, using the formula as specified in the call.
Function Dopt.augment preserves additional variables (e.g. responses) that have been added to the design design before augmenting. Note, however, that the response data are NOT used in deciding about which points to augment the design with.

The attribute run.order provides the run number in standard order (as returned from function optFederov in package **AlgDesign**) as well as the randomized actual run order. The third column is always identical to the first.

The attribute design.info is a list of various design properties, with type resolving to "Dopt", "Dopt.blocked", "Dopt.splitplot". In addition to the standard list elements (cf. design), the element quantitative is a vector of nfactor logical values or NAs, and the optional digits elements indicates the number of digits to which the data were rounded. For blocked and splitplot designs, the list contains additional information on numbers and sizes of blocks or plots, as well as the number of whole plot factors (which are always the first few factors) and split-plot factors.
The list contains a list of optimality criteria as calculated by function optFederov, see documentation there) with elements D, Dea, A and G.

(Note that replications is always 1 and repeat.only is always FALSE; these elements are only present to fulfill the formal requirements for class design. Note however, that blocked designs do in fact repeat experimental runs if nruns and blocks imply this.)

Note

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

Author(s)

Ulrike Groemping

References

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

Federov, V.V. (1972). *Theory of optimal experiments*. Academic Press, New York.

Wheeler, R.E. (2004). *Comments on algorithmic design*. Vignette accompanying package **AlgDesign**. ../../AlgDesign/doc/AlgDesign.pdf.

See Also

See also optFederov, fac.design, quad, cubic, Dopt.augment

Examples

```
 ## a full quadratic model with constraint in three quantitative factors
plan <- Dopt.design(36,factor.names=list(eins=c(100,250),zwei=c(10,30),drei=c(-25,25)),
                        nlevels=c(4,3,6),
                        formula=~quad(.),
```

```
                        constraint="!(eins>=200 & zwei==30 & drei==25)")
  plan
  cor(plan)
  y <- rnorm(36)
  r.plan <- add.response(plan, y)
  plan2 <- Dopt.augment(r.plan, m=10)
  plot(plan2)
  cor(plan2)

  ## designs with qualitative factors and blocks for
  ## an experiment on assessing stories of social situations
  ## where each subject is a block and receives a deck of 5 stories
  plan.v <- Dopt.design(480, factor.names=list(cause=c("sick","bad luck","fault"),
            consequences=c("alone","children","sick spouse"),
            gender=c("Female","Male"),
            Age=c("young","medium","old")),
            blocks=96,
            constraint="!(Age==\"young\" & consequences==\"children\")",
            formula=~.+cause:consequences+gender:consequences+Age:cause)
  ## an experiment on assessing stories of social situations
  ## with the whole block (=whole plot) factor gender of the assessor
  ##    not run for saving test time on CRAN
 ## Not run: plan.v.splitplot <- Dopt.design(480, factor.names=list(cause=c("sick","bad luck","fault"),
            consequences=c("alone","children","sick spouse"),
            gender.story=c("Female","Male"),
            Age=c("young","medium","old")),
            blocks=96,
            wholeBlockData=cbind(gender=rep(c("Female","Male"),each=48)),
            constraint="!(Age==\"young\" & consequences==\"children\")",
            formula=~.+gender+cause:consequences+gender.story:consequences+
                gender:consequences+Age:cause+gender:gender.story)
## End(Not run)
```

`lhs.design` *Functions for accessing latin hypercube sampling designs from package lhs or space-filling designs from package DiceDesign*

Description

Functions for comfortably accessing latin hypercube sampling designs from package lhs or space-filling designs from package DiceDesign, which are useful for quantitative factors with many possible levels. In particular, they can be used in computer experiments. Most of the designs are random samples.

Usage

```
lhs.design(nruns, nfactors, type="optimum", factor.names=NULL, seed=NULL, digits=NULL,
         nlevels = nruns, default.levels = c(0, 1), randomize = FALSE, ...)
lhs.augment(lhs, m=1, type="optAugment", seed=NULL, ...)
```

Arguments

nruns	number of runs in the latin hypercube sample; for type fact (a full factorial with equally-space levels), if nlevels is not separately specified, this number is taken to be the common number of levels of all factors, i.e. the resulting design will have nruns^nfactors runs; alternatively, if nlevels is separately specified as a vector of different numbers of levels, nruns can be missing or can be the correctly-specified number of runs.
nfactors	number of factors in the latin hypercube sample
type	character string indicating the type of design or augmentation method; defaults are "optimum" for lhs.design and "optAugment" for lhs.augment. Function lhs.design calls a function named typeLHS from package **lhs** (types genetic, improved, maximin, optimum, random), a function named typeDesign from package **DiceDesign** (types dmax, strauss, fact) or function runif.faure from package **DiceDesign** (type faure). Function lhs.augment calls function typeLHS from package **lhs**, where possible choices for type are augment, optSeeded, or optAugment. see the respective functions from packages **lhs** and **DiceDesign**.
seed	seed for random number generation; latin hypercube samples from package **lhs** are random samples. Specifying a seed makes the result reproducible.
factor.names	list of scale end values for each factor; names are used as variable names; the names should not be x1, x2, ..., as this would interfere with usability of standard second order analysis methods on the resulting data (link{rsmformula}); if the list is not named, the variable names are X1, X2 and so forth; the original unit cube calculated by package **lhs** (scale ends 0 and 1 for each variable) is rescaled to the values given in factor.names.
digits	digits to which the design columns are rounded; one single value (the same for all factors) or a vector of length nfactors; note that the rounding is applied after generation of the design on the actual data scale, i.e. the unit cube generated by the functions from packages **lhs** or **DiceDesign** is NOT rounded
nlevels	used for type fact only; integer number or numeric vector of nfactor integers; specifies the number of levels for each factor. If all factors have the same number of levels, the number of levels can also be specified through nruns, which is interpreted as the number of levels for type fact, if nlevels is not separately specified
default.levels	scale ends for all factors; convenient, if all factors have the same scaling that deviates from the default 0/1 scale ends.
randomize	logical that prevents randomization per default. The option has an effect for types fact and faure only. All other types are based on random design generation anyway. Note that preventing randomization is the default here, because these designs are assumed to be used mostly for computer experimentation,

where the systematics of the non-randomized design may be beneficial. For hardware experimentation, randomization should be set to TRUE!

`lhs` design generated by function `lhs.design` (class `design`, of type `lhs`

`m` integer number of additional points to add to design `lhs` (note, however, that `optSeeded` does not necessarily preserve all original runs!)

`...` additional arguments to the functions from packages **lhs** or **DiceDesign**. Refer to their documentation.
Functions for generating lhs designs: `randomLHS`, `geneticLHS`, `improvedLHS`, `maximinLHS`, `optimumLHS`, `dmaxDesign`, `straussDesign`, `runif.faure`, `factDesign`; functions for augmenting lhs designs: `augmentLHS`, `optSeededLHS`, `optAugmentLHS`)

Details

Function `lhs.design` creates a latin hypercube sample, function `lhs.augment` augments an existing latin hypercube sample (or in case of type `optSeeded` takes the existing sample as the starting point but potentially modifies it). In comparison to direct usage of package **lhs**, the functions add the possibility of recoding lhs samples to a desired range, and they embed the lhs designs into class `design`.
Range coding is based on the recoding facility from package **rsm** and the `factor.names` parameter used analogously to packages **DoE.base** and **FrF2**.

The lhs designs are useful for quantitative factors, if it is considered desirable to uniformly distribute design points over a hyperrectangular space. This is e.g. considered interesting for computer experiments, where replications of the same settings are often useless.

Supported design types are described in the documentation for packages `lhs` and `DiceDesign`.

Value

Both functions return a data frame of S3 class `design` with attributes attached. The data frame contains the experimental settings as recoded to the scale ends defined in factor.names (if given), rounded to the number of digits given in `digits` (if given). The experimental factors in the matrix `desnum` attached as attribute `desnum` contain the design in the unit cube (all experimental factors ranging from 0 to 1) as returned by packages **lhs** or **DiceDesign**.
Function `lhs.augment` preserves additional variables (e.g. responses) that have been added to the design `lhs` before augmenting. Note, however, that the response data are NOT used in deciding about which points to augment the design with.

The attribute `run.order` is not very useful for most of these designs, as there is no standard order. It therefore is present for formal reasons only and contains three identical columns of 1,2,...,nruns. For designs created with `type=fact` or `type=faure`, the standard order is the order in which package **DiceDesign** creates the design, and the actual run order may be different in case of randomization.
In case of `lhs.augment`, if the design to be augmented had been reordered before, the augmented design preserves this reorder and also the respective numbering of the design.

The attribute `design.info` is a list of various design properties, with type resolving to "lhs". In addition to the standard list elements (cf. `design`), the `subtype` element indicates the type of latin hypercube designs and possibly additional augmentations, the element `quantitative` is a vector of `nfactor` logical TRUEs, and the `digits` elements indicates the digits to which the data were rounded.
For designs created with package **DiceDesign**, special list elements from this package are also added

to `design.info`.
`randomize` is always TRUE for designs generated by random sampling, but may be FALSE for designs created with `type=fact` or `type=faure`.
`coding` provides formulae for making the designs comfortably usable with standard second order methodology implemented in package **rsm**. `replications` is always 1 and `repeat.only` is always FALSE; these elements are only present to fulfill the formal requirements for class `design`.

Note

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

Author(s)

Ulrike Groemping

References

Beachkofski, B., Grandhi, R. (2002) Improved Distributed Hypercube Sampling. American Institute of Aeronautics and Astronautics Paper 1274.

Currin C., Mitchell T., Morris M. and Ylvisaker D. (1991) Bayesian Prediction of Deterministic Functions With Applications to the Design and Analysis of Computer Experiments, *Journal of the American Statistical Association* **86**, 953–963.

Santner T.J., Williams B.J. and Notz W.I. (2003) The Design and Analysis of Computer Experiments, Springer, 121–161.

Shewry, M. C. and Wynn and H. P. (1987) Maximum entropy sampling. *Journal of Applied Statistics* **14**, 165–170.

Fang K.-T., Li R. and Sudjianto A. (2006) *Design and Modeling for Computer Experiments*, Chapman & Hall.

Stein, M. (1987) Large Sample Properties of Simulations Using Latin Hypercube Sampling. *Technometrics* **29**, 143–151.

Stocki, R. (2005) A method to improve design reliability using optimal Latin hypercube sampling. *Computer Assisted Mechanics and Engineering Sciences* **12**, 87–105.

See Also

See Also `compare` for comparing optimality criteria for various designs,
`lhs-package` and `DiceDesign-package` for the packages that do the calculations,
`FrF2`, `oa.design`, `fac.design`, `pb` for other possibilites of generating designs

Examples

```
## maximin design from package lhs
plan <- lhs.design(20,7,"maximin",digits=2)
plan
plot(plan)
cor(plan)
y <- rnorm(20)
```

```
r.plan <- add.response(plan, y)

## augmenting the design with 10 additional points, default method
plan2 <- lhs.augment(plan, m=10)
plot(plan2)
cor(plan2)

## purely random design (usually not ideal)
plan3 <- lhs.design(20,4,"random",
       factor.names=list(c(15,25), c(10,90), c(0,120), c(12,24)), digits=2)
plot(plan3)
cor(plan3)

## optimum design from package lhs (default)
plan4 <- lhs.design(20,4,"optimum",
     factor.names=list(torque=c(10,14),friction=c(25,35),
          temperature=c(-5,35),pressure=c(20,50)),digits=2)
plot(plan4)
cor(plan4)

## dmax design from package DiceDesign
## arguments range and niter_max are required
## ?dmaxDesign for more info
plan5 <- lhs.design(20,4,"dmax",
     factor.names=list(torque=c(10,14),friction=c(25,35),
          temperature=c(-5,35),pressure=c(20,50)),digits=2,
          range=0.2, niter_max=500)
plot(plan5)
cor(plan5)

## Strauss design from package DiceDesign
## argument RND is required
## ?straussDesign for more info
plan6 <- lhs.design(20,4,"strauss",
     factor.names=list(torque=c(10,14),friction=c(25,35),
          temperature=c(-5,35),pressure=c(20,50)),digits=2,
          RND = 0.2)
plot(plan6)
cor(plan6)

## full factorial design from package DiceDesign
## mini try-out version
plan7 <- lhs.design(3,4,"fact",
     factor.names=list(torque=c(10,14),friction=c(25,35),
          temperature=c(-5,35),pressure=c(20,50)),digits=2)
plot(plan7)
cor(plan7)

## Not run:

## full factorial design from package DiceDesign
## not as many different levels as runs, but only a fixed set of levels
##    caution: too many levels can easily bring down the computer
```

```
##    above design with 7 distinct levels for each factor,
##    implying 2401 runs
plan7 <- lhs.design(7,4,"fact",
     factor.names=list(torque=c(10,14),friction=c(25,35),
          temperature=c(-5,35),pressure=c(20,50)),digits=2)
plot(plan7)
cor(plan7)

## equivalent call
plan7 <- lhs.design(,4,"fact",nlevels=7,
     factor.names=list(torque=c(10,14),friction=c(25,35),
          temperature=c(-5,35),pressure=c(20,50)),digits=2)

## different number of levels for each factor
plan8 <- lhs.design(,4,"fact",nlevels=c(5,6,5,7),
     factor.names=list(torque=c(10,14),friction=c(25,35),
          temperature=c(-5,35),pressure=c(20,50)),digits=2)
plot(plan8)
cor(plan8)

## equivalent call (specifying nruns, not necessary but a good check)
plan8 <- lhs.design(1050,4,"fact",nlevels=c(5,6,5,7),
     factor.names=list(torque=c(10,14),friction=c(25,35),
          temperature=c(-5,35),pressure=c(20,50)),digits=2)

## End(Not run)
```

`optimality.criteria` *Overview of optimality criteria in experimental design packages*

Description

One function for calculating the S-optimality criterion is provided here. This help file documents this function and also describes optimality criteria from other related packages, referring to space filling optimality like the S criterion or to model-based optimality like the well-known D-criterion.

Usage

```
Scalc(design)
compare(...)
```

Arguments

`design`	matrix, often normalized to unit cube, OR design (class `design`) of type `lhs` or `Dopt`. For `design` objects, calculations are applied to the `desnum` attribute.

... two or more designs, either all of type `lhs` or all of type `Dopt`, can be compared w.r.t. some optimality criteria that are stored in their `design.info` attribute (only works with designs created by **DoE.wrapper** version 0.7 or higher)

Details

Function `Scalc` calculates the S criterion for optimality, which is employed in package **lhs** for most optimization purposes (exception: maximin designs). The criterion is the harmonic mean of all pairwise interpoint distances, and space-filling optimization tries to maximize it.

Function `eval.design` from package **AlgDesign** calculates various model-based optimality criteria:

confounding a confounding matrix of effects, if requested

determinant the k-th root of the determinant of Z'Z/N, where Z is the model matrix of the model under investigation, k is the number of columns of Z and N the number of rows; this is the quantity optimized for D-optimal designs

A the arithmetic mean of coeffient variance multipliers, i.e. the average diagonal element of the inverse of Z'Z/N, intercept included

I the average prediction variance over a space X; calculated only, if X is specified

Ge the minimax normalized variance over X; calculated only, if X is specified

Dea A lower bound on D efficiency for approximate theory designs. It is equal to exp(1-1/Ge), i.e. is also calculated only, if X is specified.

diagonality the k-th root of the ratio of the determinant of M1 divided by the product of diagonal elements of M1, where M1 is Z'Z with the column and row referring to the intercept removed, and k the number of columns of M1; if this is 1, the coefficient estimates are uncorrelated.

gmean.variances the geometric mean of normalized coeffient variance multipliers (intercept excluded), i.e. the geometric mean of the diagonal elements of the inverse of Z'Z/N, without the first element, if an intercept is in the model.

Several functions from package **DiceDesign** calculate optimality criteria regarding the space filling qualities of a design. These functions normalize the design to lie in the unit cube, if it does not yet do so. Application of these functions to designs with qualitative factors does not make sense and yields errors. The following functions are available:

`mindist` calculates the minimum distance between any pair of design points. This is the criterion which is maximized by maximin designs, i.e. should be large for a space-filling design.

For the next two distance metrics, it is helpful to define `g_i` as the minimal distance of design point `i` to any other design point.

`meshRatio` calculates the ratio of the maximum `g_i` to the minimum `g_i` (a small mesh ratio indicates a similar minimal distance for all design points).

`coverage` calculates the coefficient of variation of the `g_i`, however using the denominator `n` instead of `n-1` for the standard deviation (a large coverage indicates that the average minimal distance for of design points is large relative to their standard deviation; large values are desirable).

Finally, function `discrepancyCriteria` calculates several versions of L2 discrepancy.

Value

Function `Scalc` returns a single number: the harmonic mean of all pairwise interpoint distances is calculated, based on the matrix `design` or the desnum attribute of the design `design`. (This value should be as large as possible for a space-filling design.)

Note that the resulting S value differs from the printed optimum value by function `lhs.design` for type `optimum` in two respects: the printed optimum value is the sum of inverse distances, i.e. the denominator of the harmonic mean. choose(nruns, 2) divided by the printed final optimal value is approximately equal to the calculated S; perfect equality cannot be achieved because the underlying the printed final optimum refers to an interim latin hypercube of integers that is subsequently rescaled to the unit cube and scrambled by random numbers.

Function `compare` returns a matrix, with rows representing the criteria and columns the different designs. Apart from many of the criteria mentioned above, the determinant of the correlation matrix is shown, which should ideally be close to one for a near-orthogonal design (at least in terms of linear effects).

Note

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

Author(s)

Ulrike Groemping

References

Beachkofski, B., Grandhi, R. (2002) Improved Distributed Hypercube Sampling. American Institute of Aeronautics and Astronautics Paper 1274.

Currin C., Mitchell T., Morris M. and Ylvisaker D. (1991) Bayesian Prediction of Deterministic Functions With Applications to the Design and Analysis of Computer Experiments, *Journal of the American Statistical Association* **86**, 953–963.

Santner T.J., Williams B.J. and Notz W.I. (2003) The Design and Analysis of Computer Experiments, Springer, 121–161.

Shewry, M. C. and Wynn and H. P. (1987) Maximum entropy sampling. *Journal of Applied Statistics* **14**, 165–170.

Fang K.-T., Li R. and Sudjianto A. (2006) *Design and Modeling for Computer Experiments*, Chapman & Hall.

Stein, M. (1987) Large Sample Properties of Simulations Using Latin Hypercube Sampling. *Technometrics* **29**, 143–151.

Stocki, R. (2005) A method to improve design reliability using optimal Latin hypercube sampling. *Computer Assisted Mechanics and Engineering Sciences* **12**, 87–105.

See Also

See Also `lhs-package`, `DiceDesign-package`, `eval.design`

Examples

```
## optimum design from package lhs (default)
plan <- lhs.design(20,4,"optimum",
       factor.names=list(c(15,25), c(10,90), c(0,120), c(12,24)), digits=2)
## maximin design
plan2 <- lhs.design(20,4,"maximin",
       factor.names=list(c(15,25), c(10,90), c(0,120), c(12,24)), digits=2)
## purely random design (usually not ideal)
plan3 <- lhs.design(20,4,"random",
       factor.names=list(c(15,25), c(10,90), c(0,120), c(12,24)), digits=2)
## genetic design
plan4 <- lhs.design(20,4,"genetic",
       factor.names=list(c(15,25), c(10,90), c(0,120), c(12,24)), digits=2)
## dmax design from package DiceDesign
## arguments range and niter_max are required
## ?dmaxDesign for more info
plan5 <- lhs.design(20,4,"dmax",
     factor.names=list(torque=c(10,14),friction=c(25,35),
           temperature=c(-5,35),pressure=c(20,50)),digits=2,
           range=0.2, niter_max=500)
## Strauss design from package DiceDesign
## argument RND is required
## ?straussDesign for more info
plan6 <- lhs.design(20,4,"strauss",
     factor.names=list(torque=c(10,14),friction=c(25,35),
           temperature=c(-5,35),pressure=c(20,50)),digits=2,
           RND = 0.2)
## compare all these designs
compare(plan, plan2, plan3, plan4, plan5, plan6)
```

`rsmformula` *Functions for supporting response surface analysis with package rsm*

Description

These functions support response surface analysis with package rsm. Function rsmformula creates a model formula for use with function rsm, using the FO, TWI and PQ notation. Function code.design prepares a fractional factorial 2-level design with center points from package FrF2 or a ccd, bbd or lhs design from this package for convenient use with package rsm functionality, function decode.design reverses the coding action.

Usage

```
code.design(design)
decode.design(design)
rsmformula(design, response=NULL, factor.names=NULL,
    use.blockvar = TRUE, degree=2, coded=TRUE, ...)
```

Arguments

design	a response surface design of class design with at least one response variable and of a type derived from ccd, bbd, or lhs
response	character string specifying the response variable to be analysed (default: the first response of the design)
factor.names	character vector specifying the factors to be included (default: all experimental factors)
use.blockvar	logical indicating whether or not the block effect (if available) is to be included into the model
degree	default is 2 for a full second order model. For a first order only model, specify 1; for a model with main effects and 2-factor interactions, specify 1.5. degree corresponds to the order element of the object created by function rsm.
coded	logical indicating whether the formula is to be provided in coded names (x1, x2 etc., coded=TRUE) or original variable names (coded=FALSE)
...	reserved for future usage

Details

Function code.design rescales the variables of a design with quantitative variables according to the information stored in the coding element of the design.info attribute of design, function decode.design rescales a coded design to original units.

Function rsmformula creates a formula for use with function rsm. If this function is created in coded variable names (which is the default), it can be used in function rsm together with the coded data object created by function code.design for creating a response surface model, which can be post-processed by the utilities provided in package **rsm**, especially the [rsm:rsm]{methods} for class rsm objects and functions steepest or canonical.path.

IMPORTANT: coded vs. original units
The text below assumes that the design has been entered using the default.levels or the factor.names option to specify the factor levels in original units.
The usual steepest ascent analysis is done in coded units, i.e. if e.g. factor X1 has original units 10 (code -1 = (10-20)/10) and 30 (code +1 = (30-20)/10) and factor X2 has original units 0.1 (code -1 = (0.1 - 0.2)/0.1) and 0.3 (code +1 = (0.3 - 0.2)/0.1), an increase of 10 for a change in factor X1 from 10 to 30 is considered steeper (slope 10/2) than an increase of 9 for a change in factor X2 from 0.1 to 0.3 (slope 9/2). If this behavior is desired, usage of rsmformula with option coded=TRUE and a design generated by code.design is needed.
Otherwise, i.e. when assessment is desired in original units, the ascent for factor X2 (9/0.2) would of course be much steeper than for factor X1 (10/20) in the above example. For obtaining an assessment based on the original units, one can simply use rsmformula with option coded=FALSE and the design itself in original units in the rsm model. This only makes sense for first order models: function steepest always assesses the slope at the origin; for first order models, it does not matter where the slope is assessed. For models with order (=degree) 1.5 or 2, the steepest analysis in original units is adequate only for the exceptional case that 0 is the point of interest.

Value

Function code.design returns a coded.data object for usage with function rsm; this object can be returned to its original state by applying function decode.design.

Function `rsmformula` returns a formula with an `FO` (=first order) portion, for `degree=1.5` additionally a `TWI` (=two factor interactions, 2fis) portion, and for `degree=2` also a `PQ` (=pure quadratic) portion.

This representation of the model formula is needed for response surface analyses with package **rsm**. Per default, the formula comes in coded variable names (`x1`, `x2` etc.).

Note

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

Author(s)

Ulrike Groemping

References

Lenth, R.V. (2009). Response-Surface Methods in R, using **rsm**. *Journal of Statistical Software* **32**(7), 1-17. URLhttp://www.jstatsoft.org/v32/i07/.

Myers, R.H., Montgomery, D.C. and Anderson-Cook, C.M. (2009). *Response Surface Methodology. Process and Product Optimization Using Designed Experiments*. Wiley, New York.

See Also

See also `rsm`, `steepest`, `canonical.path`, `contour.lm`, . The `formula` method for class `design` objects creates equivalent model formulae in standard model notation.

Examples

```
## an artificial example with random response
## purely for demonstrating how the functions work together with rsm
plan <- ccd.design(5, ncenter=6,
       factor.names = list(one=c(10,30),two=c(1,5),three=c(0.1,0.9),
                           four=c(2,4),five=c(-1,1)))
set.seed(298)
plan <- add.response(plan, rnorm(38))

## coding
plan.c <- code.design(plan)
plan.c
decode.design(plan.c)

## first order analysis
   ## formulae needed for first order models:
  rsmformula(plan, degree=1)                ## coded
  rsmformula(plan, degree=1, coded=FALSE)   ## original units

  ## steepest ascent: steepness assessed in coded units,
  ## results also presented in original units
  linmod1 <- rsm(rsmformula(plan, degree=1), data=plan.c)
  summary(linmod1)
```

```
  steepest(linmod1)

  ## steepest ascent: steepness assessed in original units!!!
  ## this is different from the usual approach!!!
  ## cf. explanation in Details section
  linmod1.original <- rsm(rsmformula(plan, degree=1, coded=FALSE), data=plan)
  summary(linmod1.original)
  steepest(linmod1.original)

## second order analysis (including quadratic, degree=1.5 would omit quadratic
  ## formulae needed for second order models:
  rsmformula(plan, degree=2)               ## coded
  rsmformula(plan, degree=2, coded=FALSE)  ## original units
     ## the formulae can also be constructed analogously to the FO formulae
     ## by using SO instead of FO
     ## rsmformula returns the more detailed function because
     ##     it can be more easily modified to omit one of the effects

  ## the stationary point is not affected by using coded or original units
  ##     neither is the decision about the nature of the stationary point
  ## a subsequent canonical path analysis will however be affected,
  ##     analogously to the steepest ascent (cf. Details section)

  ## analysis in coded units
  linmod2 <- rsm(rsmformula(plan, degree=2), data=plan.c)
  summary(linmod2)
  ## analysis in original units
  linmod2.original <- rsm(rsmformula(plan, degree=2, coded=FALSE), data=plan)
  summary(linmod2.original)
  ## the contour plot may be nicer when using original units
  contour(linmod2, form=~x1*x2)
  contour(linmod2.original, form=~one*two)
  ## the canonical path is usually more reasonable in coded units
  canonical.path(linmod2)            ## coded units
  canonical.path(linmod2.original)   ## original units

  ## analogous analysis without the special formula notation of function rsm
  linmod <- rsm(rnorm.38. ~ Block.ccd + (one + two + three + four + five)^2 +
        I(one^2) + I(two^2) + I(three^2) + I(four^2) + I(five^2), data=plan)
  summary(linmod)
  contour(linmod, form=~one*two)  ## contour plot is possible
  ## steepest or canonical.path cannot be used,
  ## because the model is a conventional lm

  ## contour will not work on the convenience model
  ## lm(plan), which is otherwise identical to linmod
  ## (it will neither work on lm(formula(plan), plan))
  ## or lm(rsmformula(plan), plan)
```

Index