

Package ‘rsm’

January 21, 2012

Version 1.40

Date 2010-07-28

Title Response-surface analysis

Author Russell V. Lenth

Maintainer Russell V Lenth <russell-lenth@uiowa.edu>

Description Provides functions to generate response-surface designs, fit first- and second-order response-surface models, make surface plots, obtain the path of steepest ascent, and do canonical analysis.

License GPL-2

LazyLoad yes

LazyData yes

Repository CRAN

Date/Publication 2010-07-29 09:14:37

R topics documented:

rsm-package	2
bbd	3
ccd	4
ccd.pick	7
ChemReact	9
coded.data	10
contour.lm	12
FO	15
heli	16
model.data	17
rsm	18
steepest	20

Index	23
--------------	-----------

Description

The `rsm` package provides functions useful for designing and analyzing experiments that are done sequentially in hopes of optimizing a response surface.

The function `ccd` can generate (and randomize) a central-composite design; it allows the user to specify an aliasing or fractional blocking structure, and does a sanity check to make sure it is suitable for estimating a second-order model. The function `bbd` generates and randomizes a Box-Behnken design. The function `ccd.pick` is useful for identifying good parameter choices in central-composite designs.

The function `rsm` is an enhancement of `lm` that provides for additional analyses peculiar to response surfaces. It requires a model formula that contains a call to `F0` or `S0` to specify a first- or second-order model. Once the model is fitted, the `steepest` function may be used to obtain the direction of steepest ascent (or descent). `canonical.path` is an alternative to `steepest` for second-order response surfaces.

In RSM methods, appropriate coding of data is important not only for numerical stability, but for proper scaling of results; the function `coded.data` and its relatives facilitate this coding requirement.

Finally, a few more functions are provided that may be useful beyond response-surface applications. `contour.lm`, `persp.lm`, and `image.lm` aids in visualizing a response surface, or of any other `lm` object where a surface is fitted. `model.data` recovers the data used in a `lm` call, but unlike `model.frame`, no polynomials, factors, etc. are expanded.

For more information and examples, use `vignette("rsm")`. Additionally, `vignette("rsm-plots")` provides some illustrations of the graphics functions.

Author(s)

Russell V. Lenth

Maintainer: Russell V. Lenth <russell-lenth@uiowa.edu>

References

Box, GEP, Hunter, JS, and Hunter, WG (2005), *Statistics for Experimenters* (2nd ed.), Wiley-Interscience.

Lenth RV (2009). "Response-Surface Methods in R, Using `rsm`", *Journal of Statistical Software*, 32(7), 1–17. <http://www.jstatsoft.org/v32/i07/>.

Myers, RH, Montgomery, DC, and Anderson-Cook, CM (2009), *Response Surface Methodology* (3rd ed.), Wiley.

bbd *Generate a Box-Behnken design*

Description

This function can generate a Box-Behnken design in 3 to 7 factors, and optionally will block it orthogonally if there are 4 or 5 factors. It can also randomize the design and create a `coded.data` object.

Usage

```
bbd(k, n0 = 4, block = (k == 4 | k == 5), randomize = TRUE, coding)
```

Arguments

k	A formula, or an integer giving the number of variables. If the formula has a left-hand side, the variables named there are appended to the design and initialized to NA.
n0	Number of center points in each block.
block	Boolean specifying whether or not to block the design; or a character string (taken as TRUE) giving the desired name for the blocking factor. Only BBDs with 4 or 5 factors can be blocked. A 4-factor BBD has three orthogonal blocks, and a 5-factor BBD has two.
randomize	Boolean determining whether or not to randomize the design. If block is TRUE, each block is randomized separately.
coding	Optional list of formulas. If this is provided, a <code>coded.data</code> object is returned with the specified coding.

Details

Box-Behnken designs (BBDs) are useful designs for fitting second-order response-surface models. They use only three levels of each factor (compared with 5 for central-composite designs) and sometimes fewer runs are required than a CCD. This function uses an internal table of BBDs; it only works for 3 to 7 factors.

If k is specified as a formula, the names in the formula determine the names of the factors in the generated design. Otherwise, the names will be `x1`, `x2`, ...

Value

A `data.frame` with the generated design; or if coding is specified, a `coded.data` object. The blocking variable, if present, will be a `factor`; all other variables will be numeric.

Note

To avoid aliasing the pure-quadratic terms, you must use a positive value `n0`.

Author(s)

Russell V. Lenth

References

Lenth RV (2009). "Response-Surface Methods in R, Using rsm", *Journal of Statistical Software*, 32(7), 1–17. <http://www.jstatsoft.org/v32/i07/>.

Myers, RH, Montgomery, DC, and Anderson-Cook, CM (2009), *Response Surface Methodology* (3rd ed.), Wiley.

See Also

[ccd](#), [coded.data](#)

Examples

```
library(rsm)

### Simple 3-factor case, not randomized so structure is evident
bbd(3, randomize=FALSE)

### 5-factor BBD, divided between two plants
bbd(y1 + y2 ~ A + B + C + D + E, n0 = 5, block = "Plant")
```

ccd

Generate a central-composite design

Description

This function can generate a broad range of central-composite designs with flexible choices of replications, aliasing of predictors and fractional blocks, and choices of axis or 'star' points.

Usage

```
ccd(basis, generators, blocks = "Block", n0 = 4, alpha = "orthogonal",
    wbreps = 1, bbreps = 1, randomize = TRUE, inscribed = FALSE, coding)
```

Arguments

basis	A formula, or an integer giving the number of variables. If the formula has a left-hand side, the variables named there are appended to the design and initialized to NA.
generators	Optional formula or list of formulas to generate aliased variables
blocks	A string or a formula. If a character string, it is the name of the blocking factor; if a formula, the left-hand side is used as the name of the blocking factor, and the formula(s) on the right-hand side are used to generate separate fractional blocks.

n0	Integer or vector of two integers giving the number of center points. If only one value, this is the number of center points in each block. If two values are given, n0[1] specifies the number of center points in 'cube' blocks, and n0[2] specifies the number of center points in 'star' blocks.
alpha	If numeric, the position of the 'star' points. May also be a character string; if it matches "orthogonal", the star points are positioned to block the design orthogonally; if it matches "rotatable", the star points are chosen to make the design rotatable. The default is to generate an orthogonally blocked design. Note that a face-centered design is obtained by specifying alpha = 1.
wbrep	Number(s) of within-block replications. If this is a vector of length 2, then separate numbers are used for the 'cube' and the 'axis' blocks respectively.
bbrep	Number(s) of between-block replications (i.e., number of repeats of each block). If this is a vector of length 2, then separate numbers are used for the 'cube' and the 'axis' blocks respectively.
randomize	Boolean determining whether or not to randomize the design. Each block is randomized separately.
inscribed	Boolean value. If FALSE, the cube points are at +/- 1 and the axis points are at least 1. If TRUE, the entire design is scaled down so that the axis points are at +/- 1 and the cube points are at interior positions.
coding	Optional list of formulas. If this is provided, a <code>coded.data</code> object is returned with the specified coding.

Details

Central-composite designs (CCDs) are popular designs for use in response-surface exploration. They are blocked designs consisting of at least one 'cube' block (two-level factorial or fractional factorial, plus center points), and at least one 'star' block (points along each axis at positions $-\alpha$ and $+\alpha$), plus center points. Everything is assumed to be on a coded scale, where the cube portion of the design has values of -1 and 1 for each variable, and the center points are 0. The codings may be provided, if desired, using the coding argument.

The `basis` argument determines a basic design used to create cube blocks. For example, `basis = ~A+B+C` would generate a basic design of 8 runs. Use generators if you want additional variables; for example, `generators = c(D~A*B, E~B*C)` added to the above would generate a 5-factor design with defining relation $I = -ABD = BCE = -ACDE$.

If you want the cube points divided into fractional blocks, give the formula(s) in the `blocks` argument. For instance, suppose we have `basis = A+B+C+D+E`, `generators = F~A*C*D`, and `blocks = Day ~ c(A*B*C, C*D*E)`. Then the blocking variable will be named "Day", and the basic 32-run design (half-fraction of 6 factors) will be divided into 4 blocks of 8 runs each, based on the combination of signs of $A*B*C$ and $C*D*E$. Notice that Day will be confounded with its generators, the interaction thereof, and all aliases of these: $Day = ABC = CDE = ABDE = -BDF = -ADCF = -BCEF = -AEF$. To each of these blocks, we will add n0 (or n0[1]) center points.

A star block contains n0 (or n0[2]) center points, plus points at $\pm \alpha$ on each coordinate axis. You may specify the alpha you want, or have it chosen to achieve orthogonality of blocks (the default) or rotatability. Conditions for these criteria to hold are described in detail in references such as Myers and Montgomery (2005). In cases where there are constraints on the possible design-point

values, you may want to specify `inscribed = TRUE`. This will scale-down the design so that no coded value exceeds 1.

`wbreds` values greater than 1 cause cube points or star points to be replicated in the same block. `bbreds` values greater than 1 cause additional blocks to be added to the design. By default, the design is randomized so that the run order in each block is random. The order of appearance of the blocks is not randomized.

A couple of convenience features are added. If `basis` is an integer instead of a formula, default variable names of `x1`, `x2`, ... are used; for example, `basis = 3` is equivalent to `basis = ~x1+x2+x3`. You may specify generators or fractional blocks using the same notation. You may also give variables on the left-hand side when `basis` is a formula, and these variables are added to the returned design. For example, `Yield+ProcessTime ~ A+B+C` would generate a design in factors `A`, `B`, `C` (plus others if there are generators), and additional variables `Yield` and `ProcessTime` initialized at `NA`.

Value

A `data.frame` with the generated design; or if coding is specified, a `coded.data` object. The blocking variable will be a `factor`; all other variables will be numeric.

Note

Poor choices of generators and/or blocks can alias or partially alias some effects needed to estimate a second-order response surface. `ccd` runs a trial model on just the cube portion of the experiment, and issues a warning if there are aliased effects. The presence of center points and axis points may help mitigate this problem and make the surface estimable, but it is best to avoid designs where this aliasing occurs.

The function `ccd.pick` is available to help determine good choices for arguments to this function.

In practice, you may generate the whole design, but initially collect data only on one or more 'cube' blocks, which is good enough for estimating a first-order model. Depending on the results of analysis, we either may follow a path of steepest ascent, or continue with data from the 'star' block(s) (and possibly more 'cube' blocks) so that we can estimate a second-order model.

An alternative to a CCD when you want to go straight to second-order modeling is a Box-Behnken design, generated by `bbd`. These designs are not as various or flexible as CCDs, but they can require fewer runs.

Author(s)

Russell V. Lenth

References

- Lenth RV (2009). "Response-Surface Methods in R, Using `rsm`", *Journal of Statistical Software*, 32(7), 1–17. <http://www.jstatsoft.org/v32/i07/>.
- Myers, RH, Montgomery, DC, and Anderson-Cook, CM (2009), *Response Surface Methodology* (3rd ed.), Wiley.

See Also

[ccd.pick](#), [coded.data](#), [bbd](#)

Examples

```
library(rsm)

### Generate a standard 3-variable CCD with 12 runs in each block
des = ccd(3, n0=c(4,6), coding = list(x1 ~ (Temp - 150)/10,
                                     x2 ~ (Pres - 50)/5, x3 ~ Feedrate - 4))
decode.data(des)

### Same as above, except make the design rotatable,
### and inscribed so that no coded value exceeds 1
des2 = ccd(3, n0=c(4,6), alpha = "rotatable", inscribed = TRUE,
           coding = list(x1 ~ (Temp - 150)/10,
                         x2 ~ (Pres - 50)/5, x3 ~ Feedrate - 4))
decode.data(des2)

### Generate a 5-variable design in 2 blocks. The cube block has 16 runs
### This design will have alpha = 2; it is both orthogonal and rotatable
ccd(~ x1 + x2 + x3 + x4, x5 ~ x1 * x2 * x3 * x4, n0 = c(6,1))

### Generate a 5-variable design with 5 blocks:
### 4 blocks of 8 cube points each, and 1 block with star points
### You'll get alpha = 2.366; if you add alpha = "rot", you'll get 2.378
des = ccd(y1 + y2 ~ A + B + C + D + E, , Shift ~ c(-A*B*C, C*D*E), n0=c(2,4))
```

ccd.pick

Find a good central-composite design

Description

This function looks at all combinations of specified design parameters for central-composite designs, calculates other quantities such as the 'alpha' values for rotatability and orthogonal blocking, imposes specified restrictions, and outputs the best combinations in a order. This serves as an aid in identifying good designs. The design itself can then be generated using [ccd](#).

Usage

```
ccd.pick(k, n.c = 2^k, n0.c = 1:10, blks.c = 1, n0.s = 1:10, bbr.c = 1,
        wbr.s = 1, bbr.s = 1, best = 10, sortby = c("agreement", "N"),
        restrict)
```

Arguments

k	Number of factors in the design
n.c	Number(s) of factorial points in each cube block

n0.c	Numbers(s) of center points in each cube block
blks.c	Number(s) of cube blocks that together comprise one rep of the cube portion
n0.s	Numbers(s) of center points in each star (axis-point) block
bbr.c	Number(s) of copies of each cube block
wbr.s	Number(s) of replications of each star point within a block
bbr.s	Number(s) of copies of each star block
best	How many designs to list. Use best=NULL to list them all
sortby	String(s) containing numeric expressions that are each evaluated and used as sorting key(s). Specify sortby=NULL if no sorting is desired.
restrict	Optional string(s) containing Boolean expressions that are each evaluated. Only combinations where all expressions are TRUE are retained.

Details

A grid is created with all combinations of n.c, n0.c, ..., bbr.s. Then for each row of the grid, several additional variables are computed:

n.s The total number of axis points in each star block

N The total number of observations in the design

alpha.rot The position of axis points that make the design rotatable. Rotatability is achieved when design moment $[iiii] = 3[iijj]$ for i and j unequal.

alpha.orth The position of axis points that make the blocks mutually orthogonal. This is achieved when design moments $[ii]$ within each block are proportional to the number of observations within the block.

agreement The absolute value of the log of the ratio of alpha.rot and alpha.orth. This measures agreement between the two alphas.

If restrict is provided, each expression is evaluated and only the rows where the expression is TRUE are kept. (Regardless of restrict, rows are eliminated where there are insufficient degrees of freedom to estimate all needed effects for a second-order model.) The rows are sorted according to the expressions in sortby; the default is to sort by agreement and N, which is suitable for finding designs that are both rotatable and orthogonally blocked.

Value

A data.frame containing best or fewer rows, and variables n.c, n0.c, blks.c, n.s, n0.s, bbr.c, wbr.s, bbr.s, N, alpha.rot, and alpha.orth, as described above.

Author(s)

Russell V. Lenth

References

- Lenth RV (2009). "Response-Surface Methods in R, Using rsm", *Journal of Statistical Software*, 32(7), 1–17. <http://www.jstatsoft.org/v32/i07/>.
- Myers, RH, Montgomery, DC, and Anderson-Cook, CM (2009), *Response Surface Methodology* (3rd ed.), Wiley.

See Also[ccd](#)**Examples**

```
library(rsm)

### List CCDs in 3 factors with between 10 and 14 runs per block
ccd.pick(3, n0.c=2:6, n0.s=2:8)
### Generate the design that is listed first.
ccd(3, n0=c(6,4))

### Find designs in 5 factors containing 1, 2, or 4 cube blocks
### of 8 or 16 runs, 1 or 2 reps of each axis point,
### and no more than 70 runs altogether
ccd.pick(5, n.c=c(8,16), blks.c=c(1,2,4), wbr.s=1:2, restrict="N<=70")
```

ChemReact

Chemical Reaction Data

Description

These are uncoded data (in their original units) from a central composite design with 2 factors in 2 blocks.

Usage

ChemReact

Format

A data frame with 14 observations on the following 4 variables.

Time a numeric vector; predictor variable

Temp a numeric vector; predictor variable

Block a factor with levels B1 B2. Block B1 is a first-order design with 3 center points. Block B2 consists of axis points and 3 more center points.

Yield a numeric vector; response variable

Source

Table 7.6 of Myers, RH, Montgomery, DC, and Anderson-Cook, CM (2009), *Response Surface Methodology* (3rd ed.), Wiley.

 coded.data

Functions for coded data

Description

These functions facilitate the use of coded data in response-surface analysis.

Usage

```

coded.data(data, ..., formulas = list(...))
as.coded.data(data, ..., formulas = list(...))

decode.data(data)
val2code(X, codings)
code2val(X, codings)

## S3 method for class 'coded.data'
print(x, ...)

codings(object)
## S3 method for class 'coded.data'
codings(object)
parse.coding(form)

```

Arguments

data	A data.frame
...	Formulas for producing coded variables.
formulas	Provides an alternative way to provide the coding formulas. This is useful if you want to retrieve the codings from one dataset and use them for another.
X	A vector, matrix, or data.frame to be coded or decoded.
codings	a list of formulas (see form)
x	Coded dataset to be printed.
form	A formula as described above for ...
object	An object that contains coding formulas

Details

Typically, coding formulas are of the form $x \sim (\text{var} - \text{center}) / \text{div}$ where x and var are variable names, and center and div are numbers. The left-hand side gives the name of the coded variable, and the right-hand side should be a linear expression in the uncoded variable. This expression is evaluated at variable values of 0 and 1, then the results are used to solve for the scale center and divisor. These results are rounded to 4 digits to help ensure that zeros come out exactly.

Value

`coded.data` and `as.coded.data` return an object of class `coded.data`, which inherits from `data.frame`. `print.coded.data` is the print method for this class; it simply prints the `data.frame` and then the formulas. A named `list` of the coding formulas is saved in `attr(, "codings")`.

Use `coded.data` to convert a `data.frame` in which the variables are on their original scales. The variables named in the formulas are coded and replaced with their coded versions (and also re-named). In contrast, `as.coded.data` does not modify any of the data; it assumes the variables are already coded, and the coding information is simply added.

`decode.data` converts a dataset of class `coded.data` and returns `data.frame` containing the original variables.

`code2val` converts coded values to the original scale using the codings provided, and returns an object of the same class as `X`. `val2code` converts the other direction. When using these functions, it is essential that the names (or column names in the case of matrices) match those of the corresponding coded or uncoded variables.

`parse.coding` is mostly for internal use; it extracts and returns a `list` with two vectors: a character vector names with the two variable names, and a numeric vector `const` with the center and divisor.

`codings` is a generic function for accessing codings. `codings.coded.data` returns the list of coding formulas from a `coded.data` object. See also [codings.rsm](#).

Author(s)

Russell V. Lenth

References

Lenth RV (2009). "Response-Surface Methods in R, Using `rsm`", *Journal of Statistical Software*, 32(7), 1–17. <http://www.jstatsoft.org/v32/i07/>.

See Also

[data.frame](#)

Examples

```
library(rsm)

CR = coded.data (ChemReact, x1~(Time-85)/5, x2~(Temp-175)/5)
CR
decode.data (CR) ### will be same as ChemReact

code2val (c(x1=.5, x2=-1), codings = codings(CR))
```

contour.lm

Surface plot(s) of a fitted linear model

Description

contour and persp methods that will work with any lm object involving two or more numerical predictors.

Usage

```
## S3 method for class 'lm'
contour(x, form, at, bounds, zlim, xlabs, hook,
        plot.it=TRUE, atpos = 1, image=FALSE, img.col=terrain.colors(50), ...)
```

```
## S3 method for class 'lm'
image(x, form, at, bounds, zlim, xlabs, hook, atpos = 1, ...)
```

```
## S3 method for class 'lm'
persp(x, form, at, bounds, zlim, zlab, xlabs, col = "white",
       contours = NULL, hook, atpos = 3, theta = -25, phi = 20, r = 4,
       border = NULL, box = TRUE, ticktype = "detailed", ...)
```

Arguments

x	A lm object.
form	A formula, or a list of formulas.
at	Optional <i>named</i> list of fixed values to use for surface slices. For example, if the predictor variables are x1, x2, and x3, the contour plot of x2 versus x1 would be based on the fitted surface sliced at the x3 value specified in at; the contour plot of x3 versus x1 would be sliced at the at value for x2; etc. If not provided, at defaults to the mean value of each numeric variable, and the first level of each factor.
bounds	Optional <i>named</i> list of bounds or grid values to use for the variables having the same names. See details.
zlim	Optional zlim setting passed to <code>image</code> . If not provided, the range of values across all plotted surfaces is used.
zlab	Optional label for the vertical axis.
xlabs	Alternate labels for predictor axes (see Details).
hook	Optional list that can contain functions <code>pre.plot</code> and <code>post.plot</code> . May be used to add annotations or to re-route the graphs to separate files (see Details).
atpos	Determines where at values are displayed. A value of 1 (or 2) displays it as part of the x (or y) axis label. A value of 3 displays it as a subtitle below the plot. A value of 0 suppresses it. Any other nonzero value will cause the label to be generated but not displayed; it can be accessed via a hook function.

<code>image</code>	Set to TRUE if you want an image plot overlaid by contours.
<code>img.col</code>	Color map to use when <code>image=TRUE</code> .
<code>plot.it</code>	If TRUE, no plot is produced, just the return value.
<code>col</code>	Color or colors used for facets in the perspective plot (see details).
<code>contours</code>	If non-NULL, specifications for added contour lines in perspective plot.
<code>theta, phi</code>	Viewing angles passed to <code>persp</code> (different defaults).
<code>r</code>	Viewing distance passed to <code>persp</code> (different default).
<code>border, box</code>	Options passed to <code>persp</code> .
<code>ticktype</code>	Option passed to <code>persp</code> (different default).
<code>...</code>	Additional arguments passed to <code>contour</code> or <code>persp</code> .

Details

`form` may be a single formula or a list of formulas. A simple formula like $x_2 \sim x_1$ will produce a contour plot of the fitted regression surface for combinations of x_2 (vertical axis) and x_1 (horizontal axis). A list of several such simple formulas will produce a contour plot for each formula. A two-sided formula produces contour plots for each left-hand variable versus each right-hand variable (except when they are the same); for example, $x_1+x_3 \sim x_2+x_3$ is equivalent to `list(x1~x2, x3~x2, x1~x3)`. A one-sided formula produces contour plots for each pair of variables. For example, `~ x1+x2+x3` is equivalent to `list(x2~x1, x3~x1, x3~x2)`.

For any variables not in the `bounds` argument, a grid of 26 equally-spaced values in the observed range of that variable is used. If you specify a vector of length 2, it is interpreted as the desired range for that variable and a grid of 26 equally-spaced points is generated. If it is a vector of length 3, the first two elements are used as the range, and the third as the number of grid points. If it is a vector of length 4 or more, those values are used directly as the grid values.

By default, the predictor axes are labeled using the variable names in `form`, unless `x` is an `rsm` object, in which case the variable-coding formulas, if present, are used to generate axis labels. These labels are replaced by the entries in `xlabs` if provided. One must be careful using this to make sure that the names are mapped correctly. The entries in `xlabs` should match the respective unique variable names in `form`, after sorting them in (case-insensitive) alphabetical order. Note that if `form` is changed, it may also be necessary to change `xlabs`.

In `persp`, contour lines may be added via the `contours` argument. It may be a boolean or character value, or a list. If boolean and TRUE, default black contour lines are added to the bottom surface of the box. Character values of "top", "bottom" add black contour lines to the specified surface of the box. `contours = "colors"` puts contour lines on the bottom using the same colors as those at the same height on the surface. Other character values of `contours` are taken to be the desired color of the contour lines, plotted at the bottom. If `contours` is a list, its elements (all are optional) are used as follows:

- `z` Height where the contour lines are plotted. May be "bottom" (default), "top", or a numeric value.
- `col` Color of the lines. If not specified, they will be black. May be integer color values, color names, or "colors" to match the surface colors.
- `lwd` Line width; default is 1.

Since these functions often produce several plots, the hook argument is provided if special setups or annotations are needed for each plot. It should be a list that defines one or both of the functions `pre.plot` and `post.plot`. Both of these functions have one argument, the character vector `labs` for that plot (see Value documentation).

Additional examples and discussion of these plotting functions is available via `vignette("rsm-plots")`.

Value

A list containing information that is plotted. Each list item is itself a list with the following components:

<code>x, y</code>	The values used for the x and y axes
<code>z</code>	The matrix of fitted response values
<code>labs</code>	Character vector of length 5: Elements 1 and 2 are the x and y axis labels, elements 3 and 4 are their original variable names, and element 5 is the slice label (empty if <code>atpos</code> is 0)
<code>zlim</code>	The computed or provided <code>zlim</code> values
<code>transf</code>	(persp only) The 3D transformation for <code>trans3d</code>

Author(s)

Russell V. Lenth

References

Lenth RV (2009). "Response-Surface Methods in R, Using `rsm`", *Journal of Statistical Software*, 32(7), 1–17. <http://www.jstatsoft.org/v32/i07/>.

See Also

[contour](#)

Examples

```
library(rsm)
heli.rsm = rsm(ave ~ block + S0(x1, x2, x3, x4), data = heli)

# Plain contour plots
par(mfrow = c(2,3))
contour(heli.rsm, ~x1+x2+x3+x4, at=canonical(heli.rsm)$xs)

# Same but with image overlay, slices at origin and block 2,
# and no slice labeling
contour(heli.rsm, ~x1+x2+x3+x4, at=list(block="2"), atpos=0, image=TRUE)

# Default perspective views
persp(heli.rsm, ~x1+x2+x3+x4, at=canonical(heli.rsm)$xs)

# Same plots, souped-up with facet coloring and axis labeling
```

```

persp (heli.rsm, ~x1+x2+x3+x4, contours="col", col=rainbow(40), at=canonical(heli.rsm)$xs,
      xlabs = c("Wing area", "Wing length", "Body width", "Body length"), zlab = "Flight time")

## Not run:
### Hints for creating graphics files for use in publications...

# Save perspective plots in one PDF file (will be six pages long)
pdf(file = "heli-plots.pdf")
persp (heli.rsm, ~x1+x2+x3+x4, at=canonical(heli.rsm)$xs)
dev.off()

# Save perspective plots in six separate PNG files
png.hook = list()
png.hook$pre.plot = function(lab)
  png(file = paste(lab[3], lab[4], ".png", sep = ""))
png.hook$post.plot = function(lab)
  dev.off()
persp (heli.rsm, ~x1+x2+x3+x4, at=canonical(heli.rsm)$xs, hook = png.hook)

## End(Not run)

```

Description

Use of one of these functions in a model is how you specify the portion of the model that is to be regarded as a response-surface component.

Usage

```

FO (...)
TWI (...)
PQ (...)
SO (...)
PE (...)

```

Arguments

... The numerical predictors for the response surface, separated by commas.

Details

Use `FO()` in the model formula in `rsm` to specify a first-order response surface (i.e., a linear function) in its arguments. Use `TWI()` to generate two-way interactions, and `PQ()` to generate pure quadratic terms (squares of the `FO()` terms). A call to `SO()` creates all terms in `FO()`, `TWI()`, and `PQ()` (in that order) for those variables. However, specifying `SO()` in a model formula in `rsm` will be replaced by the explicit sum of model terms, so that the anova table shows separate sums of squares. Other

variables (such as blocks or factors) may be included in the model but should never be included in the arguments to F0 or S0.

PE is used for fitting pure-error models. It should not be used in response-surface models. This function exists primarily for use by `loftest`, but could be useful in other linear-model contexts for fitting a model that interpolates the means at each distinct combination of argument values.

Value

The functions F0, TWI, PQ, and S0 return a matrix whose columns are the required predictors.

PE returns a factor whose levels are all the distinct combinations of arguments provided to the function.

Author(s)

Russ Lenth

References

Lenth RV (2009). “Response-Surface Methods in R, Using rsm”, *Journal of Statistical Software*, 32(7), 1–17. <http://www.jstatsoft.org/v32/i07/>.

See Also

[rsm](#)

Examples

```
### See 'rsm' help for more examples

library(rsm)
### Test LOF for a regression model
ChemReact.lm = lm(Yield ~ Time*Temp, data=ChemReact, subset=1:7)
PureError.lm = update (ChemReact.lm, . ~ PE(Time,Temp))
anova (ChemReact.lm, PureError.lm)
```

heli

Paper Helicopter Data

Description

A central composite design with 4 factors in 2 blocks. These data comprise a `coded.data` object.

Usage

heli

Format

A data frame with 30 observations on the following 7 variables. Each observation reflects the results of 10 replicated flights under the same experimental conditions.

block a factor with levels 1 2. Block 1 consists of 18 observations (a full factorial plus two center points). Block 2 consists of 12 observations – 8 axis points and 4 center points.

x1 a numeric vector. Coded wing area, $x1 \sim (A - 12.4) / .6$

x2 a numeric vector. Coded length ratio, $x2 \sim (R - 2.52) / .26$

x3 a numeric vector. Coded body width, $x3 \sim (W - 1.25) / .25$

x4 a numeric vector. Coded body length, $x4 \sim (L - 2) / .5$

ave a numeric vector. Average flight time, in csec.

logSD a numeric vector. $100 * \log(\text{SD of times})$.

Source

Table 12.5 of Box, GEP, Hunter, JS, and Hunter, WG (2005), *Statistics for Experimenters* (2nd ed.), Wiley.

 model.data

Reconstruct data from a linear model

Description

Create a data frame with just the variables in the formula in a `lm` object. This is comparable to `model.matrix` or `model.frame` except that factors, polynomials, transformations, etc. are not expanded.

Usage

```
model.data(lmobj, lhs = FALSE)
```

Arguments

lmobj An object returned by `lm` or one of its relatives.

lhs Boolean indicator of whether or not to include the variable(s) on the left-hand side of the model formula.

Details

This is an easy-to-use substitute for `get_all_vars`. The formula, data, and subset arguments, if present in `lmobj`'s call, affect the result appropriately.

Value

A data frame containing each of the variables referenced in the model formula.

Author(s)

Russell V. Lenth

References

Lenth RV (2009). “Response-Surface Methods in R, Using rsm”, *Journal of Statistical Software*, 32(7), 1–17. <http://www.jstatsoft.org/v32/i07/>.

See Also

[model.matrix](#), [model.frame](#)

Examples

```
library(rsm)
trees.lm = lm(log(Volume) ~ poly(log(Girth),3), data=trees, subset=1:20)
model.frame(trees.lm)
model.data(trees.lm)
```

rsm

Response-surface regression

Description

Fit a linear model with a response-surface component, and produce appropriate analyses and summaries.

Usage

```
rsm (...)
## S3 method for class 'rsm'
summary(object, ...)
## S3 method for class 'summary.rsm'
print(x, ...)

loftest (object)
canonical (object)
## S3 method for class 'rsm'
codings(object)
```

Arguments

...	In rsm, arguments that are passed to <code>lm</code> . The model must include an <code>F0()</code> or <code>S0()</code> term to define the response-surface portion of the model. In <code>summary</code> , and <code>print</code> , additional arguments are passed to their generic methods.
object	An object of class <code>rsm</code>
x	An object produced by <code>summary</code>

Details

In `rsm`, the model formula must contain at least an `FO` term; optionally, you can add a `TWI()` term and/or a `PQ()` term as well (use the same variables in each!). For convenience, specifying `SO()` is the same as including `FO()`, `TWI()`, and `PQ()`, and is the safe, preferred way of specifying a full second-order model.

Value

`rsm` returns an `rsm` object, which is a `lm` object with additional members as follows:

<code>order</code>	The order of the model: 1 for first-order, 1.5 for first-order plus interactions, or 2 for a model that contains square terms.
<code>b</code>	The first-order response-surface coefficients.
<code>B</code>	The matrix of second-order response-surface coefficients, if present.
<code>labels</code>	Labels for the response-surface terms. These make the summary much more readable.
<code>coding</code>	Coding formulas, if provided in the <code>codings</code> argument or if the <code>data</code> argument passed to <code>lm</code> is a <code>coded.data</code> object.

`summary` is the summary method for `rsm` objects. It returns an object of class `summary.rsm`, which is an extension of the `summary.lm` class with additional list elements:

<code>sa</code>	Unit-length vector of the path of steepest ascent (first-order models only).
<code>canonical</code>	Canonical analysis (second-order models only). This is a list with elements <code>xs</code> , the stationary point, and <code>eigen</code> , the eigenanalysis of <code>B</code> (see above).
<code>lof</code>	ANOVA table including lack-of-fit test.
<code>coding</code>	Coding formulas in parent <code>rsm</code> object.

Its `print` method shows the regression summary, followed by an ANOVA and lack-of-fit test. For first-order models, it shows the direction of steepest ascent, and for second-order models, it shows the canonical analysis of the response surface.

`loftest` returns an `anova` object that tests the fitted model against a model that interpolates the means of the response-surface-variable combinations.

`canonical` returns the canonical element from `summary.rsm`.

`codings.rsm` returns a list of coding formulas if the model was fitted to `coded.data`, or `NULL` otherwise.

Author(s)

Russell V. Lenth

References

Lenth RV (2009). “Response-Surface Methods in R, Using `rsm`”, *Journal of Statistical Software*, 32(7), 1–17. <http://www.jstatsoft.org/v32/i07/>.

See Also

[FO](#), [S0](#), [lm](#), [summary](#), [coded.data](#)

Examples

```
library(rsm)
CR = coded.data (ChemReact, x1~(Time-85)/5, x2~(Temp-175)/5)

### 1st-order model, using only the first block
CR.rs1 = rsm (Yield ~ FO(x1,x2), data=CR, subset=1:7)
summary(CR.rs1)

### 2nd-order model, using both blocks
CR.rs2 = rsm (Yield ~ Block + S0(x1,x2), data=CR)
summary(CR.rs2)
```

steepest

Steepest-ascent methods for response surfaces

Description

These functions provide the path of steepest ascent (or descent) for a fitted response surface produced by [rsm](#).

Usage

```
steepest (object, dist = seq(0, 5, by = .5), descent = FALSE)
canonical.path(object, which = ifelse(descent, length(object$b), 1),
              dist = seq(-5, 5, by = 0.5), descent = FALSE)
```

Arguments

object	rsm object to be analyzed.
dist	Vector of desired distances along the path of steepest ascent or descent. In <code>steepest</code> , these must all be non-negative; in <code>canonical.path</code> , you may want both positive and negative values, which specify opposite directions from the stationary point.
descent	Set this to TRUE to obtain the path of steepest descent, or FALSE to obtain the path of steepest ascent. This value is ignored in <code>canonical.path</code> if <code>which</code> is specified.
which	Which canonical direction (eigenvector) to use.

Details

`steepest` returns the linear path of steepest ascent for first-order models, or a path obtained by ridge analysis (see Draper 1963) for second-order models. In either case, the path begins at the origin.

`canonical.path` applies only to second-order models (at least a TWI term present). It determines a linear path along one of the canonical variables, originating at the stationary point (not the origin). We need to specify which canonical variable to use. The eigenvalues obtained in the canonical analysis are always in decreasing order, so the first canonical direction will be the path of steepest ascent (or slowest descent, if all eigenvalues are negative) from the stationary point, and the last one will be the path of steepest descent (or slowest ascent, if all eigenvalues are positive). These are the defaults for which when `descent=FALSE` and `descent=TRUE` respectively.

With either function, the path in uncoded units depends on how the data are coded. Accordingly, it is important to code the predictor variables appropriately before fitting the response-surface model. See [coded.data](#) and its relatives for more information.

Value

A `data.frame` of points along the path of steepest ascent (or descent). For `steepest`, this path originates from the center of the experiment; for `canonical.path`, it starts at the stationary point. If coding information is available, the data frame also includes the uncoded values of the variables.

For first-order response surfaces, only `steepest` may be used; the path is linear in that case. For second-order surfaces, `steepest` uses ridge analysis, and the path may be curved.

Note

Take careful note of the fitted values along the outputted path (labeled `yhat`). For example, if the stationary point is a maximum (all eigenvalues negative), the fitted values from `steepest` will increase as far as the stationary point, then they will decrease as we proceed along what is now the path of slowest descent.

Author(s)

Russell V. Lenth

References

Draper, NR (1963), “Ridge analysis of response surfaces”, *Technometrics*, 5, 469–479.

Lenth RV (2009). “Response-Surface Methods in R, Using `rsm`”, *Journal of Statistical Software*, 32(7), 1–17. <http://www.jstatsoft.org/v32/i07/>.

See Also

[rsm](#), [coded.data](#)

Examples

```
library(rsm)
heli.rsm = rsm (ave ~ block + S0(x1, x2, x3, x4), data = heli)

steepest(heli.rsm)

canonical.path(heli.rsm)
```

Index

- *Topic **datasets**
 - ChemReact, 9
 - heli, 16
- *Topic **design**
 - bbd, 3
 - ccd, 4
 - ccd.pick, 7
- *Topic **hplot**
 - contour.lm, 12
- *Topic **package**
 - rsm-package, 2
- *Topic **regression**
 - coded.data, 10
 - contour.lm, 12
 - F0, 15
 - model.data, 17
 - rsm, 18
 - rsm-package, 2
 - steepest, 20
- anova, 19
- as.coded.data (coded.data), 10
- bbd, 2, 3, 6, 7
- canonical (rsm), 18
- canonical.path, 2
- canonical.path (steepest), 20
- ccd, 2, 4, 4, 7, 9
- ccd.pick, 2, 6, 7, 7
- ChemReact, 9
- code2val (coded.data), 10
- coded.data, 2–7, 10, 16, 19–21
- codings (coded.data), 10
- codings.rsm, 11
- codings.rsm (rsm), 18
- contour, 13, 14
- contour.lm, 2, 12
- data.frame, 3, 6, 11
- decode.data (coded.data), 10
- factor, 3, 6
- F0, 2, 15, 20
- get_all_vars, 17
- heli, 16
- image, 12
- image.lm, 2
- image.lm (contour.lm), 12
- list, 11
- lm, 2, 17–20
- loftest, 16
- loftest (rsm), 18
- model.data, 2, 17
- model.frame, 17, 18
- model.matrix, 17, 18
- parse.coding (coded.data), 10
- PE (F0), 15
- persp, 13
- persp.lm, 2
- persp.lm (contour.lm), 12
- PQ (F0), 15
- print.coded.data (coded.data), 10
- print.summary.rsm (rsm), 18
- rsm, 2, 13, 15, 16, 18, 20, 21
- rsm-package, 2
- S0, 2, 20
- S0 (F0), 15
- steepest, 2, 20
- summary, 20
- summary.rsm (rsm), 18
- trans3d, 14
- TWI (F0), 15
- val2code (coded.data), 10