Using the `rsm` package

Russell V. Lenth
The University of Iowa

May 5, 2009

1 Overview

The `rsm` package provides several useful functions to facilitate response-surface analysis. The primary one is the `rsm` function itself, which is an extension of `lm` but with some enhancements. In specifying a model in `rsm`, the model formula is just like in `lm`, but the response-surface portion of the model is specified using one or more of the special functions `FO` (first-order), `TWI` (two-way interactions), `PQ` (pure quadratic), or `SO` (second-order, an alias for all three of the previous functions, combined). The `summary` method for `rsm` results includes the usual regression summary (but with the coefficients compactly relabeled), an analysis of variance table with a lack-of-fit test, and additional information depending on the order of the model.

An important aspect of response-surface analysis is using an appropriate coding transformation of the data. The functions `coded.data`, `as.coded.data`, `decode.data`, `code2val`, and `val2code` facilitate these transformations; we simply provide formulas for the desired transformations. If a `coded.data` object is used in place of an ordinary `data.frame` in the call, to `rsm`, then appropriate additional output is provided in the `summary` and `steepest` outputs.

Of course, before we get to analysis, we need a good design for collecting the required data. The functions `ccd` and `bbd` are provided for generating two of the most popular classes of designs—central-composite designs (CCDs) and Box-Behnken designs (BBDs). In addition, `ccd.pick` allows one to take a quick look at various combinations of choices for CCDs and find the most suitable ones.

Auxiliary functions include `steepest` for finding a path of steepest ascent (for second-order models, this uses ridge analysis); and `contour` for obtaining a contour plot of the response surface.

2 Generating a design

Suppose that you want to experiment on a process with an aim to improving its yield. You have already done a little bit of preliminary experimentation and have identified five variables that you want to manipulate experimentally. Our plan is to develop a central-composite design, which consists of some blocks of "cube" or factorial points plus center points, and other blocks with "star" or axis points, plus center points. The cube points will be placed at positions ± 1 in coded units, and the axis points will be at $\pm\alpha$. Initially, we will only collect data on one or more of the cube blocks; then, after analyzing these data with a first-order model, we can either proceed to collect data on the other blocks and fit a second-order model, or if the fit is reasonably linear, we may want to forego the extra blocks and instead follow a path of steepest ascent.

There are a lot of choices to be made—how many center points, whether we have replications, whether the cube block(s) are fractional or full factorial, and what α to use. With response surfaces, it is desirable to have a rotatable design (where the variance of the estimated response depends only on the distance from zero). On the other hand, a CCD is built in blocks, and it is a good idea to make the block effects independent of the effects needed to estimate the response surface. To help make good choices, we can run the function `ccd.pick` to explore some possibilities. Suppose that, for practical reasons, we want no more than 16 cube points in a block, but we'd also consider ones with only 8 cube points (e.g., all $2^5 = 32$ factor combinations dividing into 4 blocks of 8 each). Since there are 5 factors, there are $2 \times 5 = 10$ axis-point positions, but it might be worth considering replicating the axis points rather than having lots of center points. And we don't want the total number of runs in the design to be too excessive—say 65 at most.

2.1 Identifying a good design

Given these considerations we will run `ccd.pick` to obtain ideas for good designs. It will compute the α values needed for rotatability and orthogonal blocking for various combinations of numbers of cube points, center points, replications, etc., and show the best few after sorting in a specified order (by default, a measure of how well the two αs agree). Here is a suitable call based on the above discussion:

```
> library(rsm)
> ccd.pick(5, n.c = c(8, 16), blks.c = c(1, 2, 4), wbr.s = 1:2,
+     restrict = "N<=65")
```

```
   n.c n0.c blks.c n.s n0.s bbr.c wbr.s bbr.s  N alpha.rot alpha.orth
1   16    6      1  10    1     1     1     1 33  2.000000   2.000000
2   16    8      1  10    2     1     1     1 36  2.000000   2.000000
3   16   10      1  10    3     1     1     1 39  2.000000   2.000000
4   16    5      2  20    1     1     2     1 63  2.000000   2.000000
5   16    8      2  10    7     1     1     1 65  2.378414   2.380476
6    8    4      4  10    7     1     1     1 65  2.378414   2.380476
7   16    1      2  10    2     1     1     1 46  2.378414   2.376354
8   16    5      2  10    5     1     1     1 57  2.378414   2.390457
9   16    4      2  10    4     1     1     1 54  2.378414   2.366432
10   8    2      4  10    4     1     1     1 54  2.378414   2.366432
```

The first one listed has a total of `N=33` runs; it has `blks.c=1` cube block with `n.c=16` cube points and `n0.c=6` center points; and star block with `n.s=10` axis points (`wbr.s=1` at each position) and `n0.s=1` center point; with these settings, the design is both rotatable and orthogonal if we use $\alpha = 2$ for the axis-point positions. The 63-run design 4 is the only one shown where the axis points are replicated; it has two 16-point cube blocks with 5 center points each, and only one center point, but replicated axis points, in the star block. This design has the pleasing feature of requiring 21 runs in each block, ad it is both rotatable and orthogonal using $\alpha = 2$. In the remaining designs, there is a slight discrepancy between the αs required for rotatability and orthogonality. Designs 5 and 6 have exactly the same number of runs, and differ only in where there are 2 blocks with 16 cube points or 4 blocks of 8 cube points. Design 10 has a slightly greater discrepancy between the αs than design 6, but fewer total runs.

2.2 Generating a CCD

Suppose that we decide to go with Design 1. To generate this design, the 16-run cube block is a half-fraction of the full 32-run design in 5 factors. This can be generated by confounding the main effect of one factor with the 4-way interaction of the others. I flipped a coin and decided to use the negative of this interaction. The `ccd` function can generate and randomize the design:

```
> ccd(~x1 + x2 + x3 + x4, x5 ~ -x1 * x2 * x3 * x4, n0 = c(6, 1))

      Block x1 x2 x3 x4 x5
C1.22     1  0  0  0  0  0
C1.6      1  1 -1  1 -1 -1
C1.2      1  1 -1 -1 -1  1
C1.1      1 -1 -1 -1 -1 -1
C1.9      1 -1 -1 -1  1  1
C1.16     1  1  1  1  1 -1
C1.15     1 -1  1  1  1  1
C1.7      1 -1  1  1 -1 -1
C1.21     1  0  0  0  0  0
C1.4      1  1  1 -1 -1 -1
C1.8      1  1  1  1 -1  1
C1.12     1  1  1 -1  1  1
C1.5      1 -1 -1  1 -1  1
C1.20     1  0  0  0  0  0
C1.14     1  1 -1  1  1  1
C1.3      1 -1  1 -1 -1  1
C1.17     1  0  0  0  0  0
C1.19     1  0  0  0  0  0
C1.13     1 -1 -1  1  1 -1
C1.18     1  0  0  0  0  0
C1.10     1  1 -1 -1  1 -1
C1.11     1 -1  1 -1  1 -1
S2.1      2 -2  0  0  0  0
S2.5      2  0  0 -2  0  0
S2.7      2  0  0  0 -2  0
S2.2      2  2  0  0  0  0
S2.11     2  0  0  0  0  0
S2.8      2  0  0  0  2  0
S2.9      2  0  0  0  0 -2
S2.4      2  0  2  0  0  0
S2.6      2  0  0  2  0  0
S2.10     2  0  0  0  0  2
S2.3      2  0 -2  0  0  0
```

By default, `ccd` chooses α for orthogonality. If we want to name the variables `x1,x2,...`, we can just give the number of variables instead of a formula in the first argument:

```
> ccd(4, x5 ~ -x1 * x2 * x3 * x4, n0 = c(6, 1))
```

To generate design 4, we use the full 32-run design, but divided into blocks two blocks of 16 runs by confounding the 5-way interaction:

```
> des4 = ccd(5, , Block ~ x1 * x2 * x3 * x4 * x5, wbr = c(1, 2),
+     n0 = c(5, 1))
```

The wbr argument specifies within-block replications for cube blocks and star blocks, respectively. There is also a bbr argument for between-block replications (i.e. additional blocks with the same factor combinations).

The ccd call for generating design 5 would be similar to the one above, but no wbr argument is needed. For design 10 (or design 6), we need to block the 32 cube points into four sets of 8, by confounding two effects with blocks:

```
> des10 = ccd(5, , Block ~ c(x1 * x2 * x3, x3 * x4 * x5), n0 = c(2,
+     4))
```

These designs, while having more total runs, may be preferred over design 1 because it is possible to run only one block (10 runs, compared with 22 runs with design 1) and still be able to estimate some first-order effects.

Because experimentation can be very expensive, it would be terrible to run the design only to find out you can't estimate all the effects. For that reason, ccd does a check to make sure we can do an analysis:

```
> bad.des = ccd(5, , Block ~ c(x1 * x2 * x3 * x4, x2 * x3 * x4 *
+     x5), n0 = c(2, 4))

Warning in ccd(~x1 + x2 + x3 + x4 + x5, , Block ~ c(x1 * x2 * x3 * x4, x2 *  :
  Some 1st or 2nd-order terms are aliased in the cube portion of this design
```

The problem here is that the generalized interaction between the two effects, $x_1x_2x_3x_4 \cdot x_2x_3x_4x_4 = x_1x_5$, is also confounded with blocks. Actually, by the time center points and axis points are added, x_1x_5 is only partially confounded; but this is still not a desirable design.

2.3 Box-Behnken designs

The bbd function is provided to generate Box-Behnken designs. These are fractional 3^k designs capable of fitting second-order models. Advantages are that they sometimes require fewer runs than a CCD, and each factor has only 3 levels instead of 5. Disadvantages are that they cannot be built-up in blocks like a CCD, and they are not rotatable. BBDs are available only for 3, 4, 5, 6, and 7 factors; and only 4- and 5-factor designs can be blocked orthogonally. Here is a BBD for 5 factors (by default, in two blocks)

```
> bbd5 = bbd(5, n0 = 1)
> nrow(bbd5)

[1] 42
```

In this case, some CCDs have fewer runs. However, the size of one block is comparable to that of the first design, and we could use it for first-order analysis.

3 Chemical reactor example

The provided dataset `ChemReact` comes from Table 7.7 of Myers and Montgomery (2002).

```
> ChemReact

    Time   Temp Block Yield
1  80.00 170.00    B1  80.5
2  80.00 180.00    B1  81.5
3  90.00 170.00    B1  82.0
4  90.00 180.00    B1  83.5
5  85.00 175.00    B1  83.9
6  85.00 175.00    B1  84.3
7  85.00 175.00    B1  84.0
8  85.00 175.00    B2  79.7
9  85.00 175.00    B2  79.8
10 85.00 175.00    B2  79.5
11 92.07 175.00    B2  78.4
12 77.93 175.00    B2  75.6
13 85.00 182.07    B2  78.5
14 85.00 167.93    B2  77.0
```

The context is that block `B1` of this data were collected first and analyzed, after which block `B2` was added and a new analysis was done. Accordingly, we will illustrate the analysis in two stages.

3.1 Coding of predictors

First, though, we need to take care of coding issues. The data are provided in their original units, and the original experiment (block `B1`) used factor settings of Time $= 85 \pm 5$ and Temp $= 175 \pm 5$, with three center points. Thus, the coded variables are $x_1 = (\text{Time} - 85)/5$ and $x_1 = (\text{Temp} - 175)/5$. Let's create a coded dataset with the appropriate codings. We do this via formulas:

```
> CR = coded.data(ChemReact, x1 ~ (Time - 85)/5, x2 ~ (Temp - 175)/5)
> CR[1:7, ]

  x1 x2 Block Yield
1 -1 -1    B1  80.5
2 -1  1    B1  81.5
3  1 -1    B1  82.0
4  1  1    B1  83.5
5  0  0    B1  83.9
6  0  0    B1  84.3
7  0  0    B1  84.0

Variable codings ...
x1 ~ (Time - 85)/5
x2 ~ (Temp - 175)/5
```

3.2 Analysis of initial block

The initial 7 runs are only good enough to estimate a first-order model. We will fit this by calling `rsm` just like we would `lm`, but use the special function `FO` (first-order response surface) in the model formula:

```
> CR.rsm1 = rsm(Yield ~ FO(x1, x2), data = CR, subset = 1:7)
> summary(CR.rsm1)

Call:
rsm(formula = Yield ~ FO(x1, x2), data = CR, subset = 1:7)

Residuals:
      1       2       3       4       5       6       7 
-0.8143 -1.0643 -1.0643 -0.8143  1.0857  1.4857  1.1857 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  82.8143     0.5472 151.346 1.14e-08 ***
x1            0.8750     0.7239   1.209    0.293    
x2            0.6250     0.7239   0.863    0.437    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 

Residual standard error: 1.448 on 4 degrees of freedom
Multiple R-squared: 0.3555,       Adjusted R-squared: 0.0333 
F-statistic: 1.103 on 2 and 4 DF,  p-value: 0.4153 

Analysis of Variance Table

Response: Yield
            Df Sum Sq Mean Sq F value  Pr(>F)
FO(x1, x2)   2 4.6250  2.3125  1.1033 0.41534
Residuals    4 8.3836  2.0959                
Lack of fit  2 8.2969  4.1485 95.7335 0.01034
Pure error   2 0.0867  0.0433                

Direction of steepest ascent (at radius 1):
       x1        x2 
0.8137335 0.5812382 

Corresponding increment in original units:
    Time     Temp 
4.068667 2.906191 
```

Note that the summary includes a lack-of-fit test, and it is significant. We can try adding two-way interactions to see if it helps:

```
> CR.rsm1.5 = update(CR.rsm1, . ~ . + TWI(x1, x2))
> summary(CR.rsm1.5)
```

```
Call:
rsm(formula = Yield ~ FO(x1, x2) + TWI(x1, x2), data = CR, subset = 1:7)

Residuals:
      1       2       3       4       5       6       7 
-0.9393 -0.9393 -0.9393 -0.9393  1.0857  1.4857  1.1857 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  82.8143     0.6295 131.560 9.68e-07 ***
x1            0.8750     0.8327   1.051    0.371    
x2            0.6250     0.8327   0.751    0.507    
x1:x2         0.1250     0.8327   0.150    0.890    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 

Residual standard error: 1.665 on 3 degrees of freedom
Multiple R-squared: 0.3603,       Adjusted R-squared: -0.2793 
F-statistic: 0.5633 on 3 and 3 DF,  p-value: 0.6755 

Analysis of Variance Table

Response: Yield
            Df Sum Sq Mean Sq  F value   Pr(>F)
FO(x1, x2)   2 4.6250  2.3125   0.8337 0.515302
TWI(x1, x2)  1 0.0625  0.0625   0.0225 0.890202
Residuals    3 8.3211  2.7737                  
Lack of fit  1 8.2344  8.2344 190.0247 0.005221
Pure error   2 0.0867  0.0433                  

Stationary point of response surface:
x1 x2 
-5 -7 

Stationary point in original units:
Time Temp 
  60  140 

Eigenanalysis:
$values
[1]  0.0625 -0.0625

$vectors
          [,1]       [,2]
[1,] 0.7071068 -0.7071068
[2,] 0.7071068  0.7071068
```

The lack of fit is still significant. Note that the `summary` output now shows a canonical analysis rather than the direction of steepest ascent, as the response surface now has second-order terms.

3.3 Analysis of combined blocks

The lack-of-fit results motivate us to collect additional runs at "star" points, plus some additional center points; these are the second block. In coded units, the data are

```
> CR[8:14, ]

       x1     x2 Block Yield
8   0.000  0.000    B2  79.7
9   0.000  0.000    B2  79.8
10  0.000  0.000    B2  79.5
11  1.414  0.000    B2  78.4
12 -1.414  0.000    B2  75.6
13  0.000  1.414    B2  78.5
14  0.000 -1.414    B2  77.0

Variable codings ...
x1 ~ (Time - 85)/5
x2 ~ (Temp - 175)/5
```

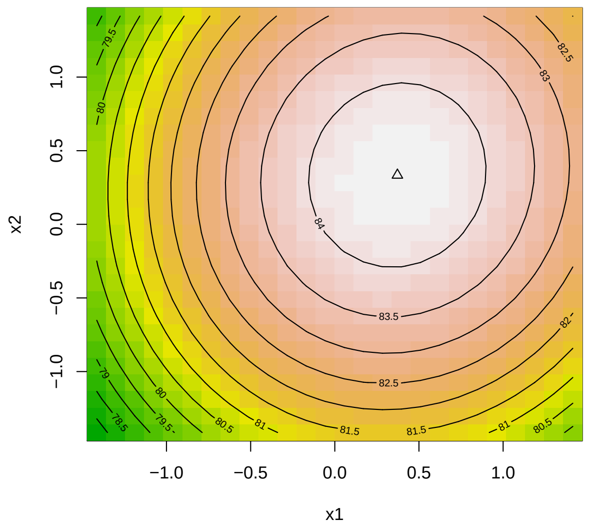
The choice of $\alpha = \sqrt{2}$ provides for rotatability, and the blocks are orthogonal as well. To do the analysis of the combined data, we should account for the block effect. We could fit a full second-order model by including `FO`, `TWI`, and `PQ` terms, but this is more easily done using `SO` which generates all three sets of variables:

```
> CR.rsm2 = rsm(Yield ~ Block + SO(x1, x2), data = CR)
> summary(CR.rsm2)

Call:
rsm(formula = Yield ~ Block + SO(x1, x2), data = CR)

Residuals:
     Min       1Q   Median       3Q      Max
-0.19543 -0.09369  0.02157  0.06153  0.20457

Coefficients:
            Estimate Std. Error  t value Pr(>|t|)
(Intercept) 84.09543    0.07963 1056.067  < 2e-16 ***
BlockB2     -4.45753    0.08723  -51.103 2.88e-10 ***
x1           0.93254    0.05770   16.162 8.44e-07 ***
x2           0.57771    0.05770   10.013 2.12e-05 ***
x1:x2        0.12500    0.08159    1.532    0.169
x1^2        -1.30856    0.06006  -21.786 1.08e-07 ***
x2^2        -0.93344    0.06006  -15.541 1.10e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 0.1632 on 7 degrees of freedom
Multiple R-squared: 0.9981,       Adjusted R-squared: 0.9964
F-statistic: 607.2 on 6 and 7 DF,  p-value: 3.811e-09

Analysis of Variance Table

Response: Yield
            Df Sum Sq Mean Sq   F value    Pr(>F)
Block        1 69.531  69.531 2611.0950 2.879e-10
FO(x1, x2)   2  9.626   4.813  180.7341 9.450e-07
TWI(x1, x2)  1  0.063   0.063    2.3470    0.1694
PQ(x1, x2)   2 17.791   8.896  334.0539 1.135e-07
Residuals    7  0.186   0.027
Lack of fit  3  0.053   0.018    0.5307    0.6851
Pure error   4  0.133   0.033

Stationary point of response surface:
       x1        x2
0.3722954 0.3343802

Stationary point in original units:
     Time      Temp
 86.86148 176.67190

Eigenanalysis:
$values
[1] -0.9233027 -1.3186949

$vectors
           [,1]       [,2]
[1,] -0.1601375 -0.9870947
[2,] -0.9870947  0.1601375
```

This model fits well. The canonical analysis reveals that the stationary point is near the center of the experiment and that both eigenvalues are negative. This indicates that the fitted surface has a maximum at Time ≈ 86.9, Temp ≈ 176.7. We may visualize the response surface using the `lm` method for `contour`, provided with this package:

```
> contour(CR.rsm2, x2 ~ x1)
> points(0.372, 0.334, pch = 2)
```



4 Helicopter example

The provided dataset `heli` is presented in Table 12.5 of Box, Hunter, and Hunter (2005). It is also a central composite design in two blocks. There are four variables and 30 observations altogether. This is a `coded.data` object already; here are a few observations:

```
> heli[1:4, ]

  block x1 x2 x3 x4 ave logSD
1     1 -1 -1 -1 -1 367    72
2     1  1 -1 -1 -1 369    72
3     1 -1  1 -1 -1 374    74
4     1  1  1 -1 -1 370    79

Variable codings ...
x1 ~ (A - 12.4)/0.6
x2 ~ (R - 2.52)/0.26
x3 ~ (W - 1.25)/0.25
x4 ~ (L - 2)/0.5
```

The response variable `ave` is the average flight time (in csec.) of four test runs each of paper helicopters made with different wing areas W, wing-length ratios R, body widths W, and body lengths L. The goal is to maximize flight time.

Like the Chemical Reaction data, the first block was analyzed first and then the star points were added. We'll skip the first part and go straight to the second-order analysis.

```
> heli.rsm = rsm(ave ~ block + SO(x1, x2, x3, x4), data = heli)
> summary(heli.rsm)

Call:
rsm(formula = ave ~ block + SO(x1, x2, x3, x4), data = heli)
```

```
Residuals:
   Min     1Q Median     3Q    Max
-3.850 -1.579 -0.175  1.925  4.200

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 372.80000    1.50638 247.481  < 2e-16 ***
block2       -2.95000    1.20779  -2.442 0.028452 *
x1           -0.08333    0.63656  -0.131 0.897707
x2            5.08333    0.63656   7.986 1.40e-06 ***
x3            0.25000    0.63656   0.393 0.700429
x4           -6.08333    0.63656  -9.557 1.63e-07 ***
x1:x2        -2.87500    0.77962  -3.688 0.002436 **
x1:x3        -3.75000    0.77962  -4.810 0.000277 ***
x1:x4         4.37500    0.77962   5.612 6.41e-05 ***
x2:x3         4.62500    0.77962   5.932 3.66e-05 ***
x2:x4        -1.50000    0.77962  -1.924 0.074926 .
x3:x4        -2.12500    0.77962  -2.726 0.016410 *
x1^2         -2.03750    0.60389  -3.374 0.004542 **
x2^2         -1.66250    0.60389  -2.753 0.015554 *
x3^2         -2.53750    0.60389  -4.202 0.000887 ***
x4^2         -0.16250    0.60389  -0.269 0.791788
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.118 on 14 degrees of freedom
Multiple R-squared: 0.9555,       Adjusted R-squared: 0.9078
F-statistic: 20.04 on 15 and 14 DF,  p-value: 6.54e-07

Analysis of Variance Table

Response: ave
                    Df  Sum Sq Mean Sq F value    Pr(>F)
block                1   16.81   16.81  1.7281  0.209786
FO(x1, x2, x3, x4)   4 1510.00  377.50 38.8175 1.965e-07
TWI(x1, x2, x3, x4)  6 1114.00  185.67 19.0917 5.355e-06
PQ(x1, x2, x3, x4)   4  282.54   70.64  7.2634  0.002201
Residuals           14  136.15    9.72
Lack of fit         10  125.40   12.54  4.6660  0.075500
Pure error           4   10.75    2.69

Stationary point of response surface:
        x1         x2         x3         x4
 0.8607107 -0.3307115 -0.8394866 -0.1161465

Stationary point in original units:
        A         R         W         L
```

```
12.916426  2.434015  1.040128  1.941927

Eigenanalysis:
$values
[1]  3.258222 -1.198324 -3.807935 -4.651963

$vectors
           [,1]       [,2]       [,3]        [,4]
[1,]  0.5177048 0.04099358  0.7608371 -0.38913772
[2,] -0.4504231 0.58176202  0.5056034  0.45059647
[3,] -0.4517232 0.37582195 -0.1219894 -0.79988915
[4,]  0.5701289 0.72015994 -0.3880860  0.07557783
```

This time, the situation is more complicated. Since the eigenvalues are of mixed sign, we have a saddle point. Here we obtain contour plots of each pair of variables, holding the other two fixed at their stationary values.

```
> par(mfrow = c(2, 3))
> contour(heli.rsm, ~x1 + x2 + x3 + x4, at = summary(heli.rsm)$canonical$xs)
```

The plots are shown in Figure 1. An important thing to note is that when the color underlay is used (as is the default), the color scale is consistent across all plots, facilitating appropriate visual comparisons.

Since we have not found a maximum, our next step might be to experiment along some path that seems promising of providing a higher response. In this particular example, the stationary point is within the experimental region, so we can regard it as reasonably well estimated. It is thus believable that the actual response function has a saddle point in the vicinity of our stationary point. The function `canonical.path`, by default, returns the path of steepest ascent each direction from the stationary point. This path is linear.

```
> canonical.path(heli.rsm)

   dist     x1     x2     x3     x4 |       A       R       W      L |    yhat
1  -5.0 -1.728  1.921  1.419 -2.967 | 11.3632 3.01946 1.60475 0.5165 | 453.627
2  -4.5 -1.469  1.696  1.193 -2.682 | 11.5186 2.96096 1.54825 0.6590 | 438.150
3  -4.0 -1.210  1.471  0.967 -2.397 | 11.6740 2.90246 1.49175 0.8015 | 424.302
4  -3.5 -0.951  1.246  0.742 -2.112 | 11.8294 2.84396 1.43550 0.9440 | 412.094
5  -3.0 -0.692  1.021  0.516 -1.827 | 11.9848 2.78546 1.37900 1.0865 | 401.504
6  -2.5 -0.434  0.795  0.290 -1.541 | 12.1396 2.72670 1.32250 1.2295 | 392.534
7  -2.0 -0.175  0.570  0.064 -1.256 | 12.2950 2.66820 1.26600 1.3720 | 385.203
8  -1.5  0.084  0.345 -0.162 -0.971 | 12.4504 2.60970 1.20950 1.5145 | 379.502
9  -1.0  0.343  0.120 -0.388 -0.686 | 12.6058 2.55120 1.15300 1.6570 | 375.429
10 -0.5  0.602 -0.105 -0.614 -0.401 | 12.7612 2.49270 1.09650 1.7995 | 372.986
11  0.0  0.861 -0.331 -0.839 -0.116 | 12.9166 2.43394 1.04025 1.9420 | 372.172
12  0.5  1.120 -0.556 -1.065  0.169 | 13.0720 2.37544 0.98375 2.0845 | 372.987
13  1.0  1.378 -0.781 -1.291  0.454 | 13.2268 2.31694 0.92725 2.2270 | 375.428
14  1.5  1.637 -1.006 -1.517  0.739 | 13.3822 2.25844 0.87075 2.3695 | 379.499
15  2.0  1.896 -1.232 -1.743  1.024 | 13.5376 2.19968 0.81425 2.5120 | 385.206
16  2.5  2.155 -1.457 -1.969  1.309 | 13.6930 2.14118 0.75775 2.6545 | 392.538
```

```
17  3.0  2.414 -1.682 -2.195  1.594 | 13.8484 2.08268 0.70125 2.7970 | 401.498
18  3.5  2.673 -1.907 -2.421  1.879 | 14.0038 2.02418 0.64475 2.9395 | 412.088
19  4.0  2.932 -2.132 -2.646  2.164 | 14.1592 1.96568 0.58850 3.0820 | 424.295
20  4.5  3.190 -2.358 -2.872  2.449 | 14.3140 1.90692 0.53200 3.2245 | 438.140
21  5.0  3.449 -2.583 -3.098  2.734 | 14.4694 1.84842 0.47550 3.3670 | 453.615
```

We should conduct additional experimental runs along this path and see where we get the most improvement in the observed response.

Had the stationary point been more distant, it would be more of an extrapolation from the range of the experiment, and thus it would not be a good starting point for further experimentation. That is, for a distant stationary point, a steepest-ascent method makes more sense. For second-order surfaces, the `steepest` function uses ridge analysis to determine an appropriate path:

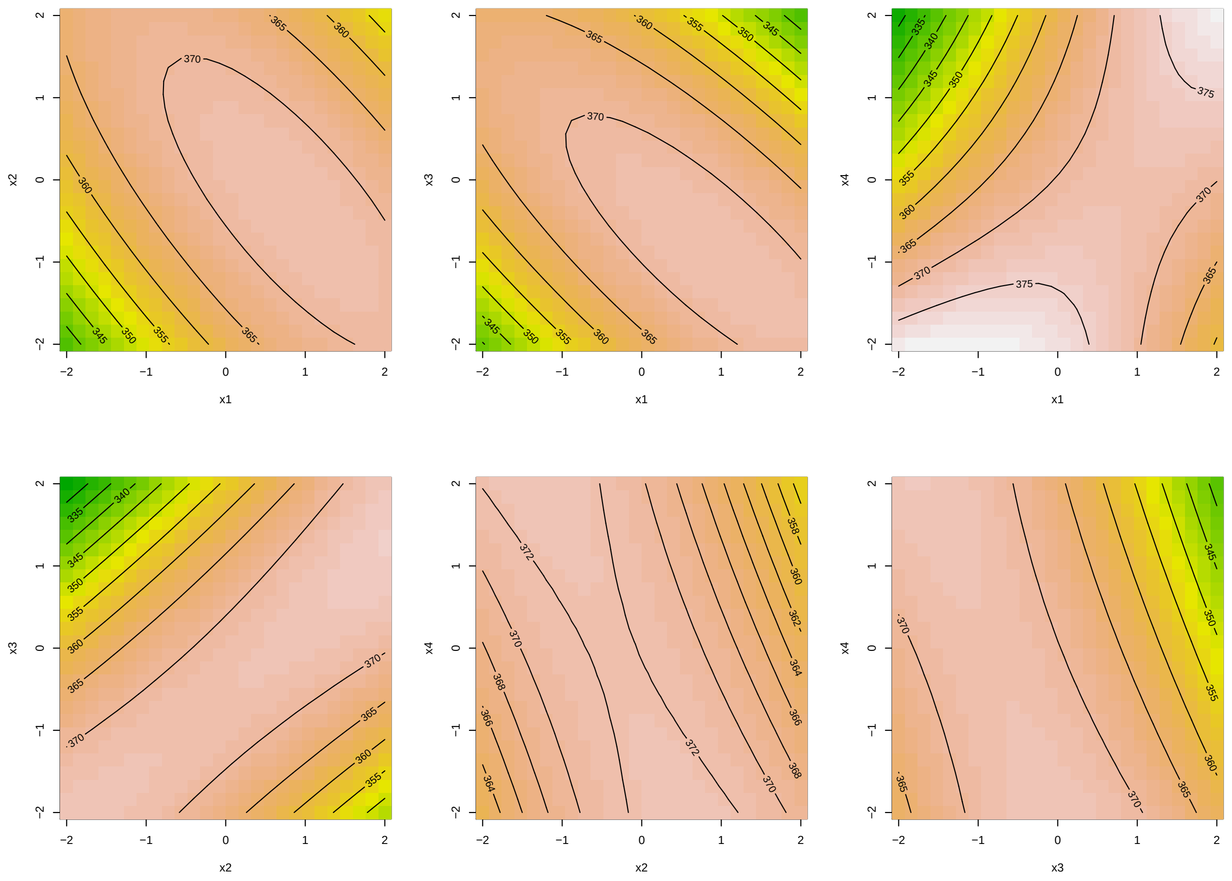


Figure 1: Contour plots for `heli` data.

```
> steepest(heli.rsm)

Path of steepest ascent from ridge analysis:
   dist     x1    x2    x3     x4 |       A       R       W      L |    yhat
1   0.0  0.000 0.000 0.000  0.000 | 12.4000 2.52000 1.25000 2.0000 | 372.800
2   0.5 -0.127 0.288 0.116 -0.371 | 12.3238 2.59488 1.27900 1.8145 | 377.106
3   1.0 -0.351 0.538 0.312 -0.700 | 12.1894 2.65988 1.32800 1.6500 | 382.675
4   1.5 -0.595 0.775 0.526 -1.009 | 12.0430 2.72150 1.38150 1.4955 | 389.783
5   2.0 -0.846 1.007 0.745 -1.309 | 11.8924 2.78182 1.43625 1.3455 | 398.485
6   2.5 -1.101 1.237 0.966 -1.605 | 11.7394 2.84162 1.49150 1.1975 | 408.819
7   3.0 -1.356 1.465 1.189 -1.897 | 11.5864 2.90090 1.54725 1.0515 | 420.740
8   3.5 -1.613 1.693 1.413 -2.188 | 11.4322 2.96018 1.60325 0.9060 | 434.322
9   4.0 -1.870 1.920 1.637 -2.477 | 11.2780 3.01920 1.65925 0.7615 | 449.497
10  4.5 -2.127 2.147 1.862 -2.766 | 11.1238 3.07822 1.71550 0.6170 | 466.323
11  5.0 -2.385 2.373 2.086 -3.054 | 10.9690 3.13698 1.77150 0.4730 | 484.750
```

This gives a path that starts at the *origin* in the coded variables, rather than the stationary point.

5 Miscellaneous notes and examples

5.1 Coded data

Use `coded.data` as shown in the Chemical reactor example to convert a dataset that has its predictors in raw units. If the dataset is already in coded units, you may embed the coding information using `as.coded.data`:

```
> dat = expand.grid(t = c(-1, 1), w = -1:1)
> dat = as.coded.data(dat, t ~ (Thickness - 3.5)/0.5, w ~ (Width -
+     12)/2)
> dat

   t  w
1 -1 -1
2  1 -1
3 -1  0
4  1  0
5 -1  1
6  1  1

Variable codings ...
t ~ (Thickness - 3.5)/0.5
w ~ (Width - 12)/2

> decode.data(dat)

  Thickness Width
1         3    10
2         4    10
3         3    12
```

```
4          4    12
5          3    14
6          4    14
```

```
> code2val(c(t = -0.5, w = 0.25), attr(dat, "codings"))
```

```
Thickness     Width
     3.25     12.50
```

The design-generation functions `ccd` and `bbd` also support coding:

```
> des = bbd(Finish ~ x1 + x2 + x3, coding = list(x1 ~ (Time - 60)/10,
+     x2 ~ (Feedrate - 2.2)/0.4, x3 ~ (Speed - 2000)/250))
> des[1:3, ]
```

```
   x1 x2 x3 Finish
9   0 -1 -1     NA
14  0  0  0     NA
6   1  0 -1     NA

Variable codings ...
x1 ~ (Time - 60)/10
x2 ~ (Feedrate - 2.2)/0.4
x3 ~ (Speed - 2000)/250
```

```
> decode.data(des[1:3, ])
```

```
   Time Feedrate Speed Finish
9    60      1.8  1750     NA
14   60      2.2  2000     NA
6    70      2.2  1750     NA
```

5.2 Contour plots

The `contour` method provided by this package works for any `lm` object, not just response surfaces. By default, it overlays the contour plot on an image plot using terrain colors. Arguments provide for the image portion to be disabled or the colors changed if desired.

To make `contour` work, it was necessary to obtain the data used by a `lm` object. The standard function `get_all_vars` does not make it very easy, and `model.frame` incorporates transformations and expands polynomials and factors. The provided function `model.data` makes it very easy to obtain just the variables included in the model formula. For example, following the first-order model for the chemical reactor example,

```
> model.data(CR.rsm1, lhs = TRUE)
```

```
  Yield x1 x2
1  80.5 -1 -1
2  81.5 -1  1
3  82.0  1 -1
4  83.5  1  1
5  83.9  0  0
6  84.3  0  0
7  84.0  0  0
```

References

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

Myers, R. H. and Montgomery, D. C. (2002), *Response Surface Methodology: Process and Product Optimization Using Designed Experiments* (2nd ed.), New York: Wiley-Interscience.

Contact information

Russell V. Lenth
Department of Statistics
The University of Iowa
Iowa City, IA, USA 52242
`russell-lenth@uiowa.edu`