Surface Plots in the rsm Package

Updated to version 2.00, 3 December 2012

Russell V. Lenth
The University of Iowa

Abstract

This is a companion to the main vignette for the **rsm** package, providing more details on how to use the `contour`, `image`, and `persp` to visualize fitted response surfaces. While designed with `rsm` objects in mind, these methods work with any `lm` object and thus provide a quick way to graph a fitted surface. Enhancements include coloring, adding contour lines to perspective plots, and hooks that provide additional annotations.

Keywords: response-surface methods, regression, contour plots, perspective plots.

1. Introduction

When a regression model is fitted using two or more continuous predictors, it is useful to present a graphical visualization of the fitted surface. To this end, the functions `contour.lm`, `persp.lm` and `image.lm` were developed and incorporated in the **rsm** package, inasmuch as surface visualization is especially important when using response-surface methods. The three functions are S3 methods for objects of class `lm`, or classes (such as `rsm`) that inherit from `lm`.

This vignette is not meant to document the functions; please refer to the help pages for details. Our goal here is to illustrate their use.

2. Models with two predictors

Consider an example using the ubiquitous `swiss` dataset that is standard in R. Let us fit a model for `Fertility` as a polynomial function of `Agriculture` and `Education`:

```
R> swiss2.lm <- lm(Fertility ~ poly(Agriculture, Education, degree=2), data=swiss)
```

The following basic calls illustrate the default results from the three functions. The results are shown in Figure 1.

```
R> library(rsm)
R> par(mfrow=c(1,3))
R> image(swiss2.lm, Education ~ Agriculture)
R> contour(swiss2.lm, Education ~ Agriculture)
R> persp(swiss2.lm, Education ~ Agriculture, zlab = "Fertility")
```

Note that we use a formula in the second argument to specify which variable goes on which

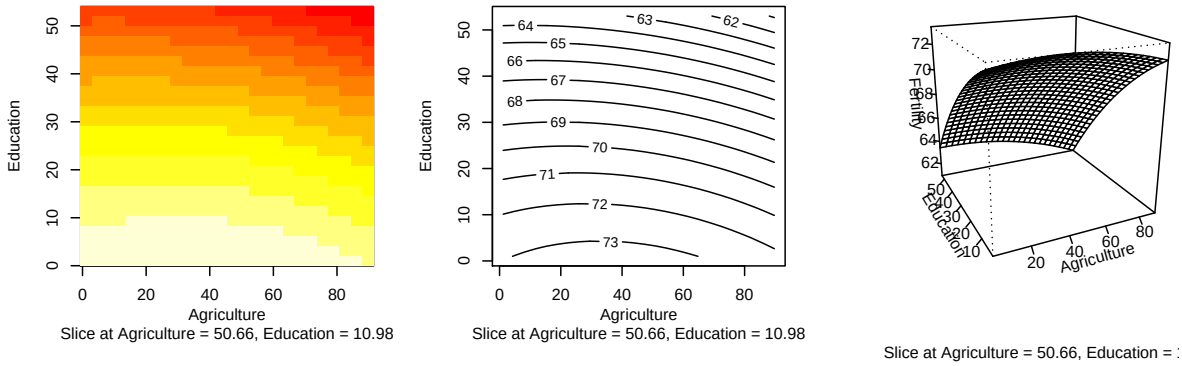

Figure 1: Basic image, contour, and perspective plots.

axis. The `persp` plot uses a different viewpoint, distance, and tick type than the default; I feel that these new defaults are better for viewing response surfaces.

Generally (as we can see in Figure 1), perspective plots are best not displayed in too small a space. It also helps to enhance them with shading, colors, or contour lines. The following call illustrates how to create an enhanced version of the perspective plot with a different point of view, shading, a different surface color, and contour lines added to the top surface of the box. We also restrict the predictor values to narrower ranges. The results are shown in Figure 2.

```
R> persp(swiss2.lm, Education ~ Agriculture, col = "blue",
+   bounds = list(Agriculture=c(20,70), Education=c(0,30)),
+   zlab = "Predicted Fertility", contours = list(z="top", col="orange"),
+   theta = -145, phi = 35, shade = 1)
```

3. Three or more predictors

When a regression model has more than two continuous predictors, some additional issues arise:

1. We can use only two predictors at a time in an image, contour, or surface plot.
2. For two given predictors, the surface plot will change depending on the values of the other predictors. (The defaults for `image`, `contour`, and `persp` is to use the average, but this can be changed.)
3. There will be more than one surface plot; it is desirable to keep the scaling and coloring consistent among all these plots. (This happens automatically in all three functions; there is no way to defeat it other than by manually plotting the retrieved surface data.)

For illustration, we will use the data from a paper-helicopter experiment described in Box, Hunter, and Hunter (2005), page 499, and provided in the **rsm** package as the dataset `heli`. The variables are coded variables x_1–x_4, which are, respectively, linear functions of wing area A, wing length ratio R, body width W, and body length L. the experiment was run in two blocks, and the response variable is `ave`, the average flight time in seconds. This dataset is

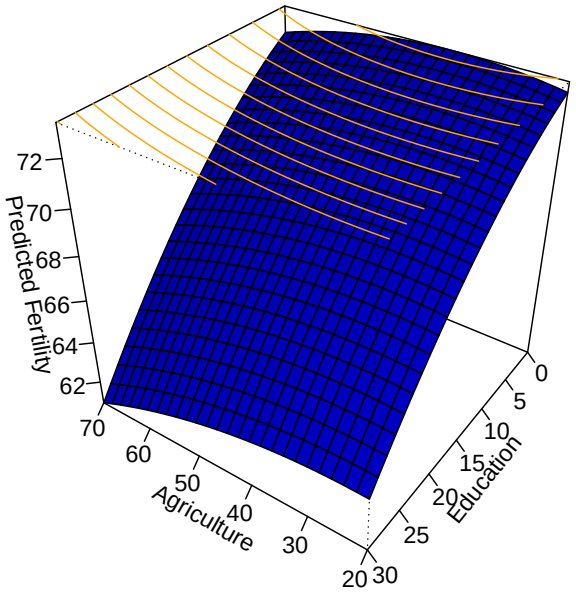

Figure 2: Enhanced perspective plot.

analyzed in more detail in the main **rsm** vignette. A second-order response-surface model for these data is obtained using

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

An `rsm` object is an extension of a `lm` object with extra response-surface-related information included. To obtain contour plots with each of the 6 possible pairs of the variables x_1–x_4, simply specify the formula `~ x1 + x2 + x3 + x4` in the call to `contour`:

```
R> par(mfrow = c(2,3))
R> contour (heli.rsm, ~ x1 + x2 + x3 + x4)
```

The result is shown in Figure 3. This dataset is an extension of `data.frame` that contains the coding information, and this information is retained in `heli.rsm`. When such coding is present, then by default, the coding formulas are used to decode the axis values x_1, x_2, x_3, x_4 to their original values A, R, W, L.

Also, when variables other than those on the coordinate axes are involved, then what is displayed is a slice of the response surface, holding the other variables fixed at certain values. By default, we use the averages of numeric predictors, and the first levels of factors. This information is incorporated as part of the x-axis label in each contour plot. In this example, we are probably more interested in the behavior of the response surface in a neighborhood of the stationary point (where the gradient is zero). We show how to do this after a little bit more discussion in the next section.

4. Annotations and hooks

Suppose in the helicopter example, we want to add some annotations to the plots. Since there

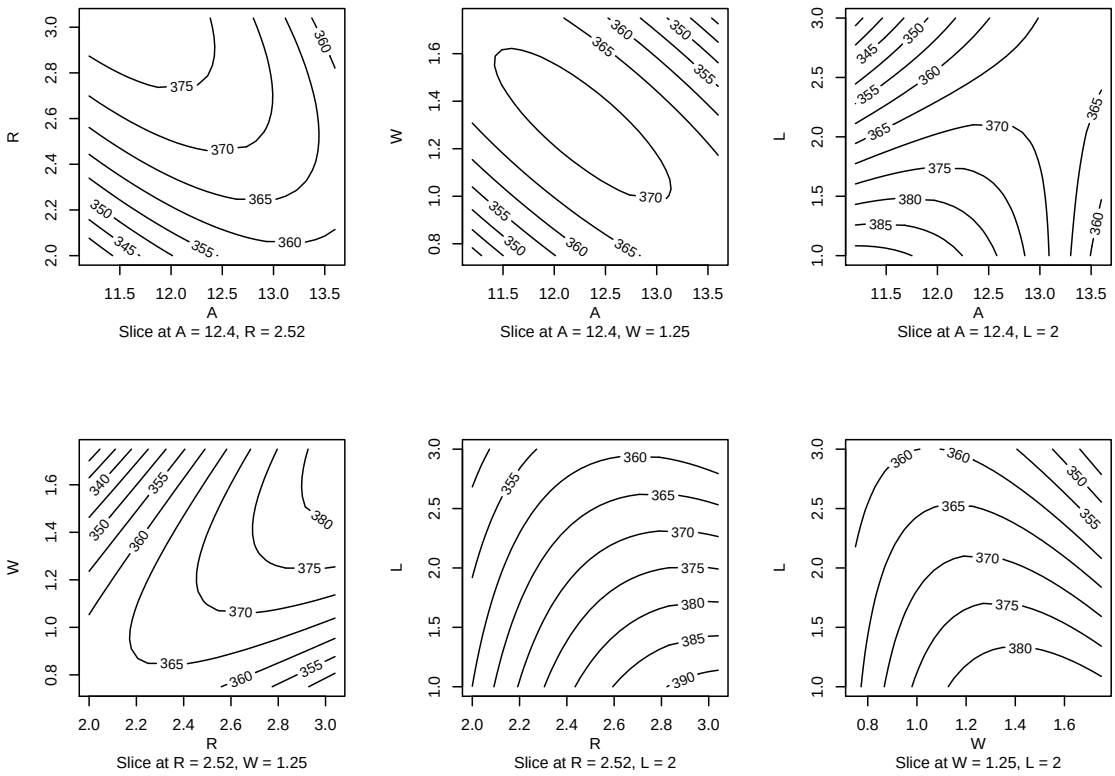

Figure 3: Default contour plots of `heli.rsm`.

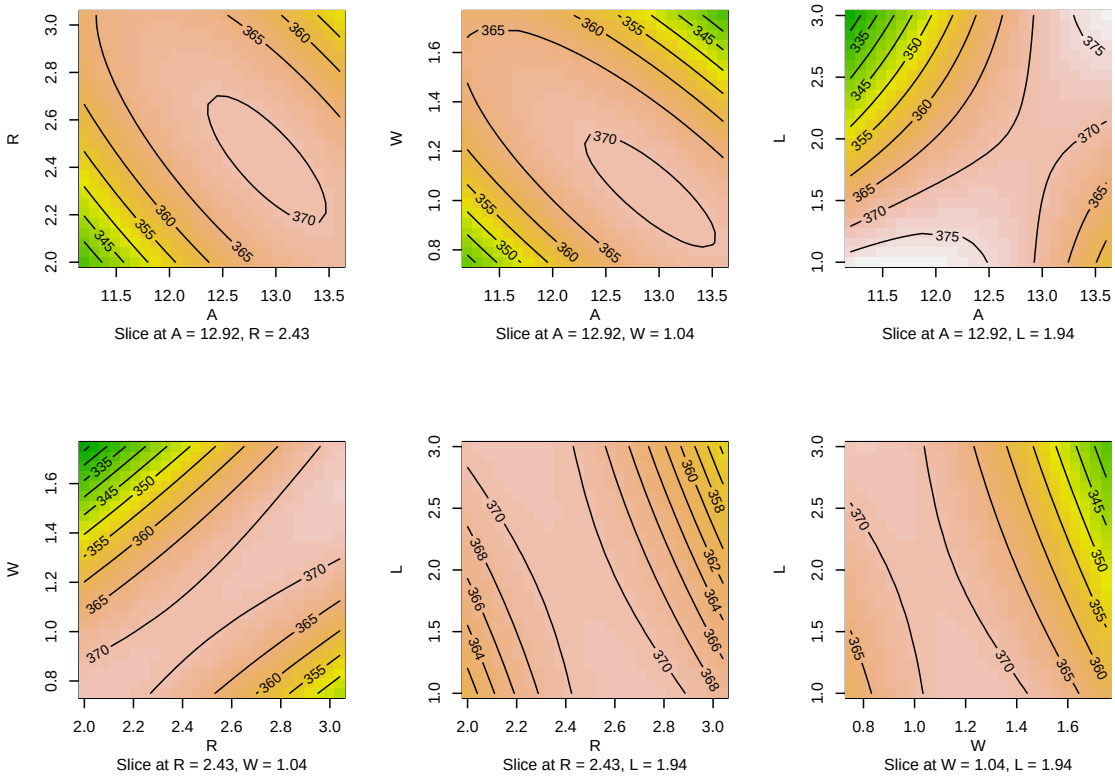

Figure 4: Enhanced contour plots of `heli.rsm`.

are several plots, we don't want to do this manually. The `contour` method for `lm` objects (as well as `image` and `persp`) allow one to specify a `hook` argument to take care of things like that. The hook should be a `list` containing function definitions for one or both of `pre.plot` and `post.plot`. Obviously, these are functions that are run just before, and just after, each plot is constructed. Each function is passed one argument, a character vector of length 4; elements 1 and 2 are the labels for the horizontal and vertical axes; elements 3 and 4 are the corresponding variable names; and element 5 is a label describing the slice being plotted.

In the following code, we set up a `post.plot` hook to plot the position of the stationary point in each graph.

```
R> xs <- canonical(heli.rsm)$xs
R> myhook <- list()
R> myhook$post.plot <- function(lab) {
+   idx <- sapply(lab[3:4], grep, names(xs))
+   points (xs[idx[1]], xs[idx[2]], pch=2, col="red")
+ }
```

The coding is a bit tedious due to the need to match elements of `xs` with the variable names. To create an enhanced contour plot, use the `at` argument to specify that we want the plots sliced at the stationary point instead of the origin, the `image` argument to enhance the plots with a background color image, and use `hook` to incorporate the above hook function. Figure 4 shows the results. Centering at the stationary point gives an entirely different view of the fitted surface than is seen in Figure 3.

```
R> par(mfrow = c(2,3))
R> contour (heli.rsm, ~ x1 + x2 + x3 + x4, image = TRUE,
+   at = xs, hook = myhook)
```

5. Saving graphs separately

An issue comes up when we don't want to squeeze all the plots into one multi-panel frame. For PostScript and PDF formats, this is easy to handle. For example:

```
R> pdf(file = "heli-cps.pdf")
R> contour (heli.rsm, ~ x1 + x2 + x3 + x4, image = TRUE, at = xs, hook = myhook)
R> dev.off()
```

With this example, the resulting file will have six pages, one per graph. We can then import, say, the fourth graph into a pdflatex source file using a command like

```
\includegraphics[width=.75\linewidth, page=4]{heli-cps.pdf}
```

If you are using **Sweave**, the device setup is already done in the background, and all you need to do is label the code chunk and import the desired graph(s).

If instead you want to create a separate file for each graph, use hooks to create files based on variable names. For example,

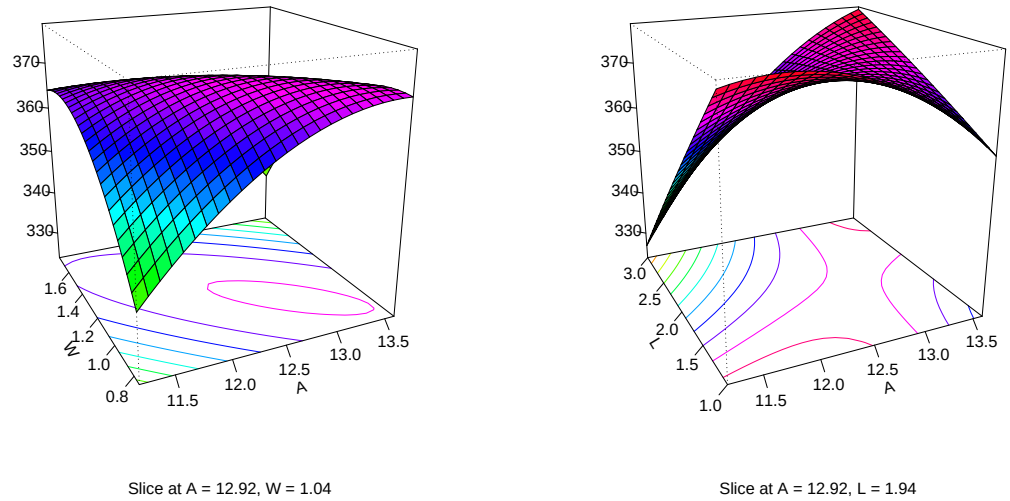

Figure 5: The first and third perspective plots with rainbow colors and contour coloring.

```
R> png.hook <- list()
R> png.hook$pre.plot <- function(lab)
+     png(file = paste(lab[3], lab[4], ".png", sep = ""))
R> png.hook$post.plot = function(lab)
+     dev.off()
R> contour (heli.rsm, ~ x1 + x2 + x3 + x4, image = TRUE, at = xs, hook = png.hook)
```

6. More on perspective plots

The `lm` method for `persp` handles its `col` argument differently than the default `persp` function. For other than a single color, it determines surface-facet colors based on the fitted response value (like is done in `image`) rather than requiring a matrix of facet colors.

Contours To add contour lines to a perspective plot, use the `contours` argument. It may be a boolean value, character value, or a list. With `contours=TRUE` or equivalently, `contours="bottom"`, contour lines are drawn on the bottom surface of the box using the default foreground color. With `contours="top"`, they are drawn at the top. Bottom contours are drawn before the surface is drawn (so they may become partially obscured), and top contours are drawn afterward. A value of `contours="colors"` will draw the contours on the bottom, using the same colors as the corresponding contour levels on the surface (as illustrated in Figure 5). Any other character value of `contours` will be taken as a color name for the contours, e.g., `contours="green"`. For more control, `contours` can be a list containing any or all of `col` (which may be either `"colors"` or a valid color), `z` (which may be `"top"`, `"bottom"`, or a numeric z value), and `lwd` (to control the width of the lines).

Colors The plots depicted in Figure 5 are two (the first and third of the perspective plots produced by

```
R> persp (heli.rsm, ~ x1 + x2 + x3 + x4, at = xs,
+   col = rainbow(50), contours = "colors")
```

The plots were saved in a six-page PDF file as described earlier, and pages 1 and 3 were imported to this document.

7. Doing it your own way

If these functions do not produce exactly the plot you want, you may still be able to save yourself a lot of work by calling `contour` with the desired object and formula(s), and `plot.it=FALSE`. The returned object is a list of data for each plot—the x and y values, the z matrix, the range of z across all plots, and axis labels.

References

Box GEP, Hunter WG, Hunter JS (2005). *Statistics for Experimenters: An Introduction to Design, Data Analysis, and Model Building*. 2nd edition. John Wiley & Sons, New York.

Affiliation:

Russell V. Lenth
Department of Statistics
The University of Iowa
Iowa City, IA 52242, United States of America
E-mail: russell-lenth@uiowa.edu
URL: http://www.stat.uiowa.edu/~rlenth/