3-D Graphics in R

Luke Keele
Ohio State University

December 6, 2005

Three dimensional graphics may not be widely used in published work, but in political methodology, three dimensional plots have a number of uses. The plotting of likelihoods and response surfaces from Monte Carlo experiments are examples of where three dimensional plots are quite useful.

The options for software are not extensive, however. Excel can produce such graphs but the quality of the plots is not excellent. One would assume that `R` would be a natural choice for producing such plots given `R`'s generally excellent graphing capabilities. And it is a good choice, but to get publication quality 3-D graphics requires more work than typically required when plotting in `R`. What I hope to do, here, is save others the time it took me to figure out how to produce quality graphics with the `wireframe` command.

The standard 3-D plot command in `R` is `persp`. While one can produce good looking plots with `persp`, it has a couple of key deficiencies when it comes to producing publication quality plots. First, the axes do not have the full functionality typical in most `R` plots. One cannot adjust the tick marks and must rely on automatic axis creation. Second, one cannot use plotmath characters as axis labels. Therefore, one is stuck with "Alpha" instead of "α" for axis labels. While these may seem like small deficiencies, they are problematic when producing publication quality plots.

The next option in `R` is to use the `wireframe` command from the `lattice` package. The `lattice` library is an `R` implementation of the `trellis` graphics package in S-plus. While the `lattice` package can do a variety of plots, it is primarily designed to plot relationships between three variables. Typically, this is done with the `xyplot` command to generate scatterplots between two variables conditional on the values of a third variable. The focus on this type of plot is unfortunate, since it means that surface plots as produced in `wireframe` tend to be overlooked in the documentation. In fact, the documentation for all the `lattice` graphics commands is sparse, and for `wireframe` is particularly thin.[1] This is doubly unfortunate since `lattice\trellis` commands tend to be complex.

To start with, let's discuss how one's data needs to be set up. For a single 3-D plot, one needs three vectors of data. Two which define the values of the x and y axes and one for the quantity of interest to be plotted. For users familiar with `persp`, this is an important difference. In `persp`, the response variable needs to be in matrix not vector form. To use `wireframe`, the response variable must be in vector form. Fortunately, for many users it is probably more convenient to have data in vector form.

As an example, I use data from a Monte Carlo experiment on misspecification in an

[1] One can use the freely available `trellis` manual when using `lattice`, but it too only gives sparse consideration to the the `wireframe` command.

OLS regression. I want to plot the bias as a function of two parameters: α and ϕ.

I need my data in R to take the following form:

```
> ols[1:20,]
      a       bias  phi
1  0.05 0.01716063 0.05
2  0.10 0.04543059 0.05
3  0.15 0.06780803 0.05
4  0.20 0.09976802 0.05
5  0.25 0.12303042 0.05
6  0.30 0.15644003 0.05
7  0.35 0.19504754 0.05
8  0.40 0.22869483 0.05
9  0.45 0.27677437 0.05
10 0.50 0.32982011 0.05
11 0.55 0.39086068 0.05
12 0.60 0.45282179 0.05
13 0.65 0.54175071 0.05
14 0.70 0.61998347 0.05
15 0.75 0.72936067 0.05
16 0.05 0.02583256 0.10
17 0.10 0.04417769 0.10
18 0.15 0.06806339 0.10
19 0.20 0.09310019 0.10
20 0.25 0.12514266 0.10
```

The two parameters, α and ϕ, both run from 0.05 to 0.75 in increments of 0.05. If one only has the response variable in vector format, the `expand.grid` command can be used to create the parameter vectors. See the R documentation or Fox (2002) for examples. Notice that I have used close increments between the values of α and ϕ. This will make the surface plot smoother. Next, I need to load the `lattice` package:

```
> library(lattice)
```

To change the background color of the plotting area to white from the default grey, I use a high level plotting command:

```
> trellis.par.set(theme = col.whitebg())
```

The command `trellis.par.set` can be used to set a large number of graphing parameters. A call to `trellis.par.get()` will list at least 50 default graphing parameters and their values. Now let's produce a basic 3-D plot with the `wireframe` command:

```
> fig1 <- wireframe(bias ~ a * phi,
+         data = ols, screen =
+         list(z = -245, x = -75),
+         xlab = expression(paste(alpha)),
+         ylab = expression(paste(phi)),
+         zlab = "Bias",
+         zlim = range(seq(0.0, 0.85,
+         by=0.20)))
```

Many of the normal plotting options such as `xlab` and `ylab` work as expected. One obvious addition to these commands is `zlab` and `zlim`. These two options work as one would expect them to. Notice also that I can include `plotmath` characters. One extra option available for all the label commands in the `rot` option. This rotates the axis labels. This option is only available if you are using R version 2.1.0 or newer. To rotate the z-axis label you would use the following: `zlab = list(label = "Bias", rot = 90)`. This is useful when the z-axis label is very long, or to rotate the x- and y-axis labels to be parallel to the axis when the plot is at an angle. Other changes to the axis labels can be made in a similar fashion. See the next plot for an example.

Let's start to clean the plot up. First, you probably noticed the box that goes around the

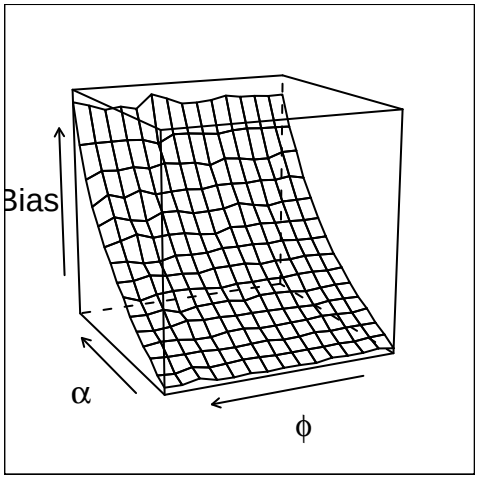


Figure 1: Basic Plot

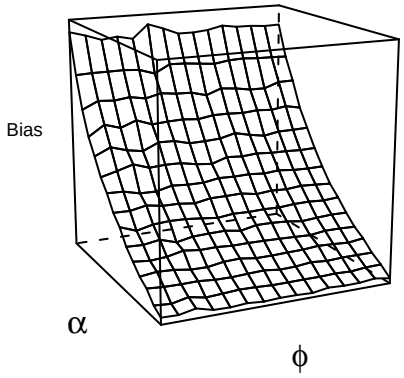


Figure 2: Removing The Border and Changing Axis Features

outside of the entire plot. Removing it is quite easy, but finding the command to do so is not. It is controlled by a high level command for the axes. To remove it, I use `trellis.par.set`. I also clean up the axis labeling in the next example:

```
> trellis.par.set("axis.line",
+ list(col="transparent"))

> fig2 <- wireframe(bias ~ a * phi,
+      data = ols,
+      screen = list(z = -245, x = -75),
+      xlab = expression(paste(alpha)),
+      ylab = expression(paste(phi)),
+      zlab = list(label = "Bias",
+      font = 1, cex = 0.60),
+      zlim = range(seq(0.0, 0.85,
+      by=0.20)))
```

The call to `trellis.par.set` removes the box around the plot. Notice now that the axis tick marks are also gone. This is because the command I used to remove the border is an axis command. We'll get the tick marks back in a moment. Let's now work through the more important options available in `wireframe`.

The most important option is probably `screen`, which controls the rotation of the plot. The first value is the z value which rotates the plot in either clockwise or counter-clockwise directions around a vertical axis. A value of 0 situates the plot so it sits perpendicular to the viewer. 180 or -180 will rotate the plot 180 degrees from the 0 angle in either direction. I find that a corner view is often best, but it will depend on the values of the surface you are plotting. For example, compare a head on view in Figure 3 to the viewing angle in Figure 2:

```
> fig3 <- wireframe(bias ~ a * phi,
+        data = ols,
+        screen = list(z = 0, x = -75),
+        xlab = expression(paste(alpha)),
+        ylab = expression(paste(phi)),
+        zlab = list(label = "Bias",
+        font = 1, cex = 0.60),
+        zlim = range(seq(0.0, 0.85,
+        by=0.20)))
```

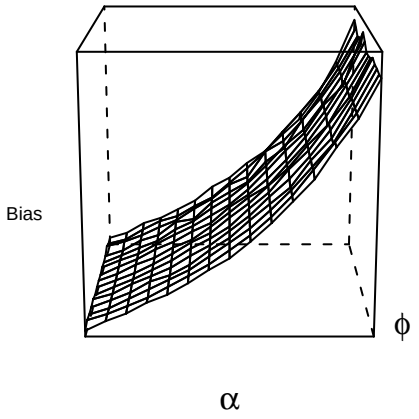


Figure 3: Rotating the plot, Z set to 0

to Figure 2, where the viewing angle of the plot is set such that we view the surface from the corner. In general, I find values around 50, 150, etc. to be quite useful. In Figure 2, for example, I use -245. Generally some experimentation is required depending on the plot.[2] The x option rotates the plot about a horizontal axis. The value 90 or -90, makes it flat so to speak as in Figure 4. Values of 0 or 180 will will show it from the top or the bottom, but this usually isn't very useful. I find a value of -75, a slight angle, is a good viewing angle but this may vary from plot to plot.

[2]There is an R library called RGL to make graphs interactive including the ability to rotate the plot with the mouse.

```
> fig4 <- wireframe(bias ~ a * phi,
+        data = ols,
+        screen = list(z = -245, x = 90),
+        xlab = expression(paste(alpha)),
+        ylab = expression(paste(phi)),
+        zlab = list(label = "Bias",
+        font = 1, cex = 0.60),
+        zlim = range(seq(0.0, 0.85,
+        by=0.20)))
```

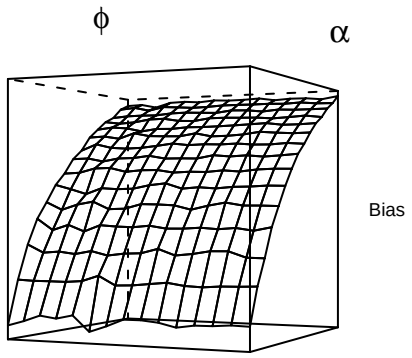


Figure 4: The Plot At A Level Viewing Angle

Another important option command is `scales`. It controls the tick marks and the labels for the tick marks. Table 1 contains a list of the options for the scales command. The

list in Table 1 is not exhaustive but contains the options I find most useful.

Table 1: Parameters of Scale Option

cex	Font Size: 1 is Default
font	1 Plain, 2 Bold, 3 Italic, 4 Bold Italic
tck	Tick Length: 1 Default
col	Color: "black" Draws tick marks
arrows	Draws Arrows Instead of Axis
draw	Draw Axis

In the next plot, I use the `scales` option to put the tick marks back on the plot by adding "black" as a color, and I adjust the size of the tick mark labels and the font.

```
> fig5 <- wireframe(bias ~ a * phi,
+        data = ols,
+        scales = list(arrows=FALSE,
+        cex= .45, col = "black",
+        font = 3, tck),
+        screen = list(z = -245, x = -75),
+        xlab = expression(paste(alpha)),
+        ylab = expression(paste(phi)),
+        zlab = list(label = "Bias",
+        font = 1, cex = 0.60), zlim =
+        range(seq(0.0, 0.85, by=0.20)))
```

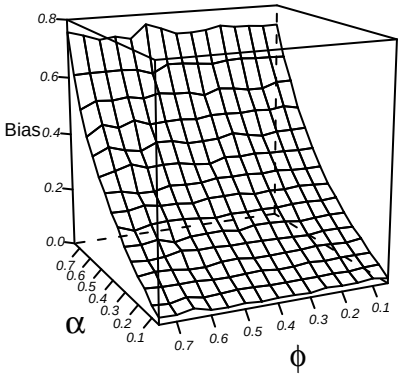


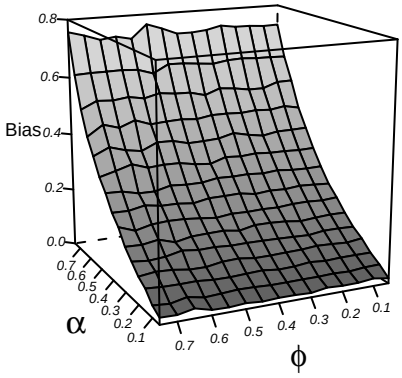
Figure 5: Controlling the Axis

Next, we'll cover some options for filling in the surface. There are two options for the surface. The first is to use the `drape` option. This adds a grid and colors to the surface. The color is chosen by `R` and is set to shades of red and orange. The next plot demonstrates this default coloring.

```
> fig6 <- wireframe(bias ~ a * phi,
=        data = ols,
+        scales = list(arrows=FALSE,
+        cex= .45, col = "black", font = 3),
+        drape = TRUE, colorkey = FALSE,
+        screen = list(z = -245, x = -75),
+        xlab = expression(paste(alpha)),
+        ylab = expression(paste(phi)),
+        zlab = list(label = "Bias",
+        font = 1, cex = 0.60),
+        zlim = range(seq(0.0, 0.85,
+        by=0.20)))
```

This looks fine when printed in black and white. The `drape` coloring doesn't accentuate the actual surface much.

To add colors, first add the option `shade = TRUE`, this option automatically overrides the `drape` option even if it is included. The default color is a rainbow like pattern. One may add a `shade.colors` function to produce any color. Figure 7 adds grey coloring. The code




```
+        font = 3, tck = 1),
+        screen = list(z = -245, x = -75),
+        colorkey = FALSE,
+        xlab = expression(paste(alpha)),
+        ylab = expression(paste(phi)),
+        zlab = list(label = "Bias",
+        font = 1, cex = 0.60),
+        shade=TRUE,
+        light.source= c(0,10,10),
+        zlim = range(seq(0.0, 0.85,
+        by=0.20)),
+        shade.colors = function(irr, ref,
+        height, w = 0.4)
+        grey(w * irr + (1 - w) *
+        (1 - (1 - ref)^0.4)))
```

Figure 6: Adding Color To The Surface Area

for `shade.colors` is complicated but to adjust the shade of grey, one need not understand all the settings with the exception of the w parameter[3] I have it set to 0.5, which produces a medium-light grey. Shifting it downward toward 0 darkens the grey and increasing it lightens the grey. Using this `shade.colors` function yields a nice looking surface when printed in black and white, as can be seen below. The grey color scheme tends to emphasize how smooth or rough the surface is. Setting `shade` to FALSE produces a grid surface with no color.

```
> fig7 <- wireframe(bias ~ a * phi,
+        data = ols,
+        scales = list(arrows=FALSE,
+        cex= 0.45, col = "black",
```

[3]See the documentation for panel.cloud for a description of the parameters in this color function.

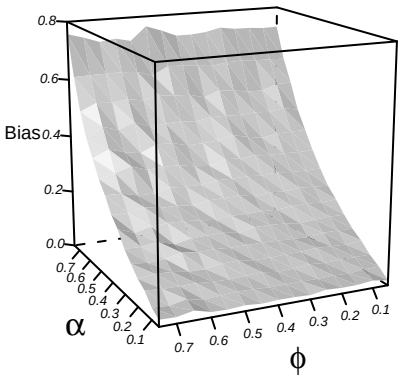


Figure 7: Greyscale Coloring For the Surface Area

There is one last option that is of only limited usefulness for a grey surface. The `lightsource` option controls the direction of the lighting in the plot, and is accompanied by

a vector of Cartesian coordinates as seen in the above examples. With a grey surface changing these values only serves to darken different parts of the surface. Changing the first coordinate darkens the entire surface, changing the second value darkens one end and changing the third darkens the other end. One can use the `lightsource` option to further accentuate the roughness of the surface, if so desired.

Another useful command is the `zoom` option. The default is set to 0.75. Decreasing it will shrink the plot. Increasing it much above 1 makes the plot bigger than is probably ever useful unless for a poster display. One can also adjust the aspect ratio. The `aspect= c(1,1)` option takes two values. When set to (1,1) the plot is exactly square. Decreasing the first value stretches the length of the plot, while decreasing the second value stretches the height of the plot. In the next example, I stretch the length of the plot.

```
> fig8 <- wireframe(bias ~ a * phi,
+         data = ols, scales =
+         list(arrows=FALSE, cex= 0.45,
+         col = "black", font = 3, tck = 1),
+         screen = list(z = -245, x = -75),
+         colorkey = FALSE,
+         xlab = expression(paste(alpha)),
+         ylab = expression(paste(phi)),
+         zlab = list(label = "Bias",
+         font = 1, cex = 0.60),
+         shade=TRUE,
+         light.source= c(0,10,10),
+         zlim = range(seq(0.0, 0.85,
+         by=0.20)),
+         shade.colors = function(irr, ref,
+         height, w = 0.4)
+         grey(w * irr + (1 - w) *
+         (1 - (1 - ref)^0.4)),
+         aspect = c(1, 0.65))
```

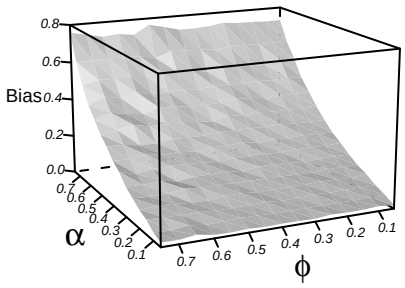


Figure 8: Changing the Aspect Ratio

Finally, I turn to getting your `lattice` plot out of `R` and into your favorite word processing program. If using the GUI version of `R` in Windows, one can always right-click and save on the plot device and save the plot as a Windows metafile or postscript file. This works fine but doesn't give the user any control over the size of the plot other than eyeballing it. Moreover, this method isn't useful for those who don't run the GUI Windows version of `R`. When using `lattice` graphics one can't use the usual graphics devices such as `postscript`. In truth, they will still work but the results are unsatisfying. Instead the `lattice` package has its own graphics device called `trellis.device`. Fortunately, it works much like the other graphics devices. Below is a code snippet the contains an example:

```
> trellis.device(postscript,
+    file="3dfig8.eps",
+    onefile = FALSE, paper = "special",
+    horizontal = FALSE,
+    width = 4, height = 4)
> trellis.par.set("axis.line",
+    list(col="transparent"))
> print(fig8)
> dev.off()
```

In the example, above, I print a postscript file, but `trellis.device` supports Windows metafiles, Mac, png, pdf, x11, and a few others. The documentation doesn't say this, but all the options from the standard graphics devices work as normal. When `trellis.device` is invoked the high level options are reset, which is why they must be included as in the example above. These can be set within the `trellis.device` command but I find it easier to keep track of the changes by making separate calls after the device is turned on. Finally, I use the `print` command to send a copy of the plot to my working directory. The `print` command has added functionality in that it can be used to print more than one plot in a grid layout. While useful in theory, trying to fit more than one or two plots on a page usually looks bad.

One can have much finer control of `wireframe` plots, but this requires exhaustive reading of the dense and unhelpful documentation. The purpose, here, has been to give the reader enough to produce publication quality plots without being driven to distraction by the `lattice` documentation.

References

Fox, John. 2002. *An R and S-Plus Companion to Applied Regression.* Thousand Oaks: Sage.