3D Graphing in R

Graphing in general is a useful tool for visualizing data, and relationships within data. 3D graphing in particular is useful for the visualization of multivariate data. R has several options for creating and exploring 3D graphs. The Lattice package offers two functions for 3D graphing, one that creates scatter plots and another that creates surfaces. These can be used to look for patterns in data or visualize planes and surfaces. Manipulating Lattice graphs is somewhat similar to working with the regular graphing functions in R, but also involves a unique set of parameter options.

One drawback to the Lattice package is that it creates non-interactive graphs. Another option is to look at R data in GGobi, which allows for interactive 3D graphing. Users can manually rotate 3D graphs and mark data points of interest. This can be a more efficient way to visually explore a dataset in multiple dimensions.

This how-to manual will provide a basic introduction to using 3D graphing in R to explore multivariate data. There are other options available; see the references list for some useful sources.

Packages and Functions Relevant to 3D Graphing

Non-interactive graphing: Lattice package

One helpful package to use when doing 3D graphing in R is the Lattice package. To load this package, use the following command:

```
library(lattice)
```

To see some example of graphics options in Lattice, use `demo(lattice)`.

Instead of using `par()` to adjust general graphing parameter options, Lattice graphing options can be adjusted with `trellis.par.set()`. Use `trellis.par.get()` to see the parameters of the current settings (this will be the default setting if there is not a graph open). The parameters are organized by groups each containing a specific list of settings. For example, `trellis.par.get("par.main.text")` will return the current group of settings for the main label of the graph. Two main ways to adjust the parameter settings are by using `trellis.par.set()` or by using `par.settings` within the command to make a graph. The first method will adjust the settings for all subsequent graphs, whereas the second method is specific to the graph it modifies. Here is an example of these two methods:

```
trellis.par.set("par.main.text", list(col="red", font=4))
cloud(x, data, par.settings=list(par.main.text, list(col="red", font=4 )))
```

The 3D graphing functions in Lattice also have specific options that can be manipulated within that function.

The cloud function

Within Lattice, cloud can be used to create 3D scatter plots (Figure 1). The general form of cloud is:

```
cloud(x, data, ...
```

x is the data that will be graphed, in the form of `z~x+y | g1*g2*...` where g1, g2, etc. are factors used for conditioning. For example, if one column in a dataset is a factor with three levels, entering this column as `g1` will produce three graphs, one for each factor level.

data is to specify the dataset
Beyond x and data, other commands can be specified within `cloud()` that format the graph:

- **groups** — specifies any groups in the data that will be identified by different colors in the plot. For example, if a dataset has a column for sex, entering `groups=sex` would color code male and female in the scatter plot.
- **auto.key** — if set to ‘true’ with groups specified, will provide a key with the graph.
- **xlab, ylab, zlab** — use a character string to label each axis of the graph. For example `xlab="x-axis"
- **subset** — identifies subset of the data to be included in the plot. For example `subset=c(20:25)` would only graph data from rows 20 through 25 of the dataset.
- **xlim, ylim, zlim** — defines the numeric range of the x-axis, y-axis, and z-axis. For example `xlim=c(50,100)` will focus the plot on x values between 50 and 100.
- **type** — specifies the type of plot: “p” for points, “l” for line, “b” for both points and lines, and “h” for histogram (a line goes from each point to the X-Y plane).
- **aspect** — specifies the aspect ratios for y-axis:x-axis and z-axis:x-axis in a numeric vector of length 2, for example `aspect=c(1:2,2)`.
- **layout** — a numeric vector of length 3 that specifies the number of columns, rows, and pages in which to display multiple graph panels (for example if conditioning function by factors g1, g2, etc.).
- **panel.aspect** — specifies the aspect ratio for the panel in which the entire plot appears.
- **panel** — used to create the display panel; see `?panel.cloud` for default settings.
- **scales** — list that describes the scales of the axes. By default, cloud draws arrows along each axis; this can be changed to tick marks along each axis with `scale=list(arrows=FALSE)`. Alternatively, `draw=FALSE` will result in a graph without arrows or tick marks along the axes.

`Panel.cloud()` shows the default panel functions for cloud, and more options for changing the appearance of the graph. These commands can be included in the `cloud()` command when creating a graph. One especially useful command is `screen`, which is a list indicating how much to rotate the graph around each axis, allowing different views. Another useful command for adjusting view is `distance`. This is specified by a value between 0 and 1 and changes the amount of perspective.

**The wireframe function**

Wireframe is the other function within Lattice that creates 3D graphs, and is used to graph surfaces. Wireframe is the Trellis version of `persp()`. See figure 2 for a common example used to demonstrate this type of graph. The general form of the wireframe function is similar to that for cloud:

```r
wireframe(x, data, ...)
```

Many of the same parameter setting options as described for `cloud()` also apply to `wireframe()`. Some settings unique to wireframe include:

- **drape** — used to indicate whether the surface is draped in color, specified with true or false
- **shade** — used to indicate whether the surface is colors and illuminated from a single light source; setting this to true overrides drape
- light.source – vector of length 3 indicating the location of the light source
- colorkey – used to indicate whether a color key should be included on the graph; specified with true or false

Interactive graphing: GGobi

(Note: the following instructions for using GGobi are specific to Macs; Ggobi won’t be required for the final question that goes with this manual)

GGobi allows for interactive graphing, including 3D and higher dimension plots. The rggobi package allows for the graphing of R datasets in GGobi. To load this package, use the following command:

```
install.packages(rggobi, dependencies=TRUE)
```

Once rggobi has been installed, open the X11 terminal. In R, enter `library(rggobi)`. Once the rggobi package is loaded in R, the command `ggobi()` can be used to open a GGobi window, which will open in X11. Putting the name of a dataset in the parentheses of the `ggobi()` command will open that particular dataset in a GGobi window. This command can also reference a file (see `?ggobi`).

Try looking at an R dataset in GGobi. Once X11 is open and GGobi is loaded, enter `ggobi(iris)`. A scatter plot window and a GGobi console window will open in X11. In the GGobi console window, under the View menu, select Rotation to look the data in 3D. The options in this window allow for rotating the scatter plot to look at the data from various views (Figure 3).

Next, under the Interaction menu in the console window, select Brush. The options in this window allow for highlighting specific points on the scatter plot (Figures 4 and 5).

Try graphing a similar 3D scatterplot in R:

```
cloud(Petal.Length~Sepal.Length+Sepal.Width, data=iris)
```

GGobi allows for easy manipulation of 3D graphs and examination of data. Patterns in the data are more easily identified compared to using the Lattice package.

Example

Run the following code to create the example dataset:

```
subject <- 1:10
age <- rep(c(19,21,24,27,31,34,37,39), times=10)
trial <- 1:10
task <- 1:2
data.sim <- expand.grid(subject=subject, trial=trial, task=task)
data.sim <- cbind(data.sim[,1], age, data.sim[,2:3])
names(data.sim) <- c("subject","age","trial","task")
data.sim$age <- as.numeric(data.sim$age)
data.sim$trial <- as.numeric(data.sim$trial)
data.sim$task <- as.numeric(data.sim$task)
RT <- ((data.sim$age+data.sim$trial^1.25)*data.sim$task)*100
logRT <- log(RT)
data.sim <- cbind(data.sim, RT, logRT)
```

This is simulated data for 10 subjects in an experiment. There are two tasks in the experiment, each with 10 trials. The dependent variable is reaction time, and there is an additional variable with the log of each reaction time.

First we can use `cloud()` to look at the data.

```
cloud(RT~trial+age, data=data.sim)
```

This basic graph is not very informative. Changing the view might be useful, and adding values to the axes will also help.
Cloud(RT~trial+age, data=data.sim, scales=list(arrows=FALSE),
screen=list(x=30, y=30, z=30))

This view starts to show that the data might be grouped in some way. We can plot the
two tasks in a way to better see any difference,
cloud(RT~trial+age, groups=task, data=data.sim,
scales=list(arrows=FALSE), screen=list(x=30, z=30, y=30),
pch=c(19, 25), col=c('blue', 'red'))

We can also look at these two groupings in separate panels.
cloud(RT~trial+age | task, data=data.sim, scales=list(arrows=FALSE),
screen=list(x=30, z=30, y=30))

These data form a plane for each task type. At this point, wireframe is useful for
visualizing a 3D surface. Again we can start by looking at the data as a whole with a
simple wireframe graph.
wireframe(RT~trial+age, data=data.sim)

The whole dataset can be represented by one plane, but we know that there is difference
between tasks, so let’s account for that.
wireframe(RT~trial+age, groups=task, data=data.sim)

Wireframe is not especially strong at overlaying separate planes. Instead we can look at
the two tasks in different panels again.
wireframe(RT~trial+age | task, data=data.sim)

And we can do a few things to make these more informative and nicer looking.
wireframe(RT~trial+age | task, data=data.sim,
scales=list(arrows=FALSE), shade=TRUE, main="RT by Age and Trial
for Task 1 and Task 2")

References
http://addictedtor.free.fr/graphiques/allgraph.php

Many examples of various graphs that can be made in R; includes R code.

GGobi website - http://www.ggobi.org/

Information about downloading and using GGobi.


Very helpful and clearly written book that includes many examples.


This particular chapter is all about the Lattice package.
Figure 1. Example of the cloud function: cloud(Volume~Girth+Height,data=trees)
Figure 2. Example of the wireframe function: wireframe(volcano)
Figure 3. GGobi console window for rotation of a 3D scatter plot.
Figure 4. GGobi console window for the Brush function.
Figure 5. GGobi scatter plot of the R dataset iris, after using Brush to highlight some data points.