Advanced Plotting

Topics

1. Customizing base graphics
2. grid graphics
3. Lattice graphics
4. ggplot2

Once you’ve managed to make a few graphs in R, you’ll likely quickly find yourself dissatisfied with their appearance. The default options are, in my opinion, horrible, so you want to start changing simple things like line thickness, axis placement, white space, and legends. You’ll also likely find that these things seem impossibly convoluted and difficult at first, particularly within the base graphics framework. You have several options. You could learn base graphics in detail (write really long “par” statements), or you could learn one or more of the other graphics systems people have developed precisely because they don’t like messing with the base system. We’ll discuss both of these options and I’ll give quick overviews of them as well as pointers to resources so you can decide for yourself. There isn’t one right approach here—it is just a matter of figuring out what works for you. I’ll admit my bias up front—I like grid graphics combined with base graphics! It gives me absolute control and the syntax and structure make sense to me. (It is also what the other advanced graphing packages we’ll discuss—lattice and ggplot2—are built on). Check out the R graphics page for a detailed run-down of all things graphical in R.

Customizing base graphics

There are many ways that you can change the default appearance of a plot produced using base (or “traditional”) graphics (i.e. something produced by `plot()`, `barplot()`, etc.) These can roughly be split into two categories: messing with spacing in and around figures, and messing with the stuff you actually draw. I will emphasize that the best (only?) way to learn how to do this is to read the help files for individual commands and try it out. That being said, I’ll go over some of the overarching bits to hopefully make you feel more comfortable about where to start. For more detail, I recommend *R Graphics* [1].

The first thing you have to understand is that when you plot something, you are adding to a preexisting canvas and the details of what that canvas looks like are specified by `par()`. Commands like `plot()` and `points()` can also change graphics settings but they only do so temporarily, whereas changes via `par()` persist until you issue another `par()` command to change them. You can see what all the `par()` settings are simply by issuing the empty command, which will return a long list of graphics parameter settings:

```r
> par()
```

Because you will be messing with many of these parameter settings, it is useful to be able to save your original settings to restore the defaults without having to restart R. To save the current settings:

```r
> oldpar <- par(no.readonly = TRUE)
> ... [lots of changes to par]
> par <- oldpar
```

Some of the graphical parameters cannot be changed (“read only”), so the `no.readonly = TRUE` bit simply passes all of the `par()` settings that aren’t read only. The second command simply resets `par()` parameters to the values you saved with the first.

Spacing

Rather than trying to go through all of the possible `par` spacing settings, I’ll go over how to modify the general layout of the different areas of a graphics device. I’ll discuss this in detail on the board, with reference to Figure 1. The graphics device is further divided into `figure` and `plot` regions, and the placement of these regions on the page is determined by `margins`. The “outer margin” (`om_`) specifies the distance between the
edge of the device and the edge of the figure region whereas the “inner margin” (ma_) specifies the distance between the edge of the figure and the edge of the plot. The third letter in each of these margin specifications tells you the units: lines of text (oma, mar), or inches (omi, mai). These are always specified as vectors of 4 values (c(0,1,1,2)) for the bottom, left, top, and right margins, respectively, so order is important!

The alternative approach to setting margins is to specify the normalized location of regions relative to their containers. The normalized location of the “inner region” is changed with omd. The inner region is the same as the figure region for a single plot whereas it is the container holding all figure regions for multipanel plots. The parameters fig and plt are the corresponding commands for changing the normalized location of the figure and plot regions. By normalized, it means that the values can go from 0 (left side of region) to 1 (right side of region). These normalized locations are also specified with 4 element vectors, but the elements have a very different meaning/order than do the vector specifications for margins. The first two elements specify the distance from the left side of the containing region of the left and right sides of the embedded region whereas the last two elements specify the distance from the bottom of the containing region of the bottom and top of the embedded region. This sounds very obtuse, but is fairly straightforward: if you specify par(plt=c(.1,.9,.2,.8)), the left side of the plot will be 10% from the left side of the figure region, and the right side of the plot will be 90% from the left side of the figure region. Similarly, the bottom of the plot will be 20% up from the bottom of the figure region and the top of the plot will be 80% up from the bottom of the figure region.

grid graphics

Both lattice and ggplot2 (see below) are built on grid graphics, which is a “low-level” system for producing graphics contained in the package grid. I like it because it provides a really nice and intuitive way to construct plots from the get-go. In combination with base graphics, you can construct a scaffolding for a figure, then use “high-level” plotting functions to do the actual plotting. Chapter 5 of R Graphics is a good introduction to grid graphics and is available here.

lattice

This package was developed with ideas about data visualization and ease of use in mind, and the focus is primarily on presentation of multivariate data. Unfortunately, I find the syntax and logic obtuse, so hardly ever use it. However, many people love it to the point of using it almost exclusively, and others use it primarily for data exploration. I’ll only briefly go over the basics of lattice and refer you to references that should help you learn it further should you so desire. The main positive is that you can quickly produce multipanel graphs.

Figure 1: General overview of regions and spacing (margins) for base graphics. Note that the blue section is the “plot” region.
to visualize your data in meaningful ways. The main negatives (to me) are syntax. Rather than run through lattice in detail, I'd like to suggest some resources for those who wish to try it for themselves. Chapter 8 of A Beginner's Guide to R [2] is an introduction to the lattice package and can be downloaded here (UW library access to the pdf). Chapter 4 of R Graphics [1] is a nice introduction to lattice and can be downloaded (text and data) here.

**ggplot2**

This is a package developed by Hadley Wickham (important R and stats person) that attempts to adopt the “grammar of graphics” in the way we plot graphics. I’ve played with it some and am more and more convinced that it is an excellent direction to go. The syntax is intuitive and the default display is pretty nice (though a bit cluttered for my taste). Wickham has written an accompanying text (appropriately titled ggplot2, and his website is really nice and useful). The first chapter is available here and is enough to get you started.

**More practice with plotting**

We’ll use A Beginner’s Guide to R [2], Chapter 7 “Graphing Tools” to get more practice with plotting.

→ Download Graphing Tools and the data and scripts.

As before, follow along in the script (BGR_chap7.R) and play with the code to learn how to use these alternative plotting techniques.

**Homework**

Do ONE of the following by Thursday, Oct. 24, 2013:

1. Submit a script with answers to the Exercises in BGR chapter 7 (and only answers to the exercises – nothing else).

2. Apply these alternative plotting techniques to your data to refine a previous plot or make a new one. Submit the script and data necessary to make the plot with any questions you have embedded in the script.
