Creating maps in R

Allan Hicks
allan.hicks@noaa.gov
11/28/2012

Introduction to maps in R

R is a useful tool to visualize data on maps, and a few packages are available to make it easy to plot data onto maps. Displaying data on a map gives the reader a sense of place, distance, and relation. For example, you might display locations where data were collected and associate some data at those locations (Figure 1).

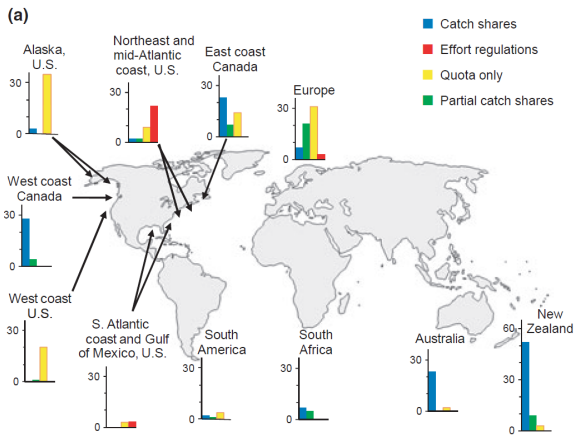

Figure 1: Figure from Melnychuk et al 2001 (Fish and Fisheries)

Or, you may show patterns across a locale (Figure 2)

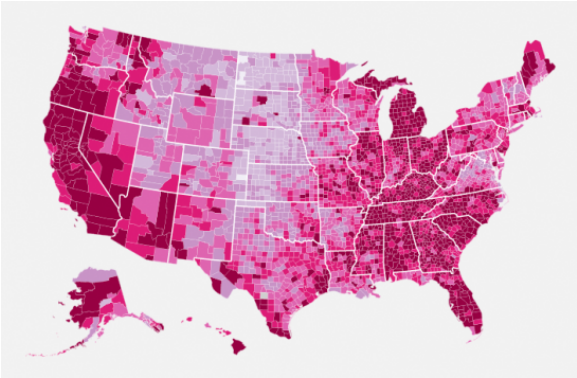

Figure 2: Chloropleth map of unemployment rate http://blog.revolutionanalytics.com/2009/11/choropleth-map-r-challenge.html

R makes it easy to produce maps like these, but it is not a substitution for Graphic Information Systems (GIS). GIS is much more than producing maps. But, R has the ability to draw decent maps for the purpose of visualization and possibly publication.

Projections

Before discussing the map packages in R, it is important to realize there are different projections. Earth is a sphere, at least the last time I checked, and one way to show a map of the world is to use a globe. However, packing a globe containing detail of a hiking trail into a daypack is a bit prohibitive. Therefore, cartographers project the surface of the earth onto a 2-dimensional surface. This causes distortion, and different projections have been developed to preserve some properties. For example, there are equal area projections (preserve area), conformal projections (preserve angles), equidistant (preserve distance from some standard point or line), and more. Projections are also classified by the mechanism in which the globe is conceptually projected, such as cylindrical, conical, and azimuthal. Projections can become quite complicated (Figure 3) and there are many different opinions on which projection is best. I will not explore projections further

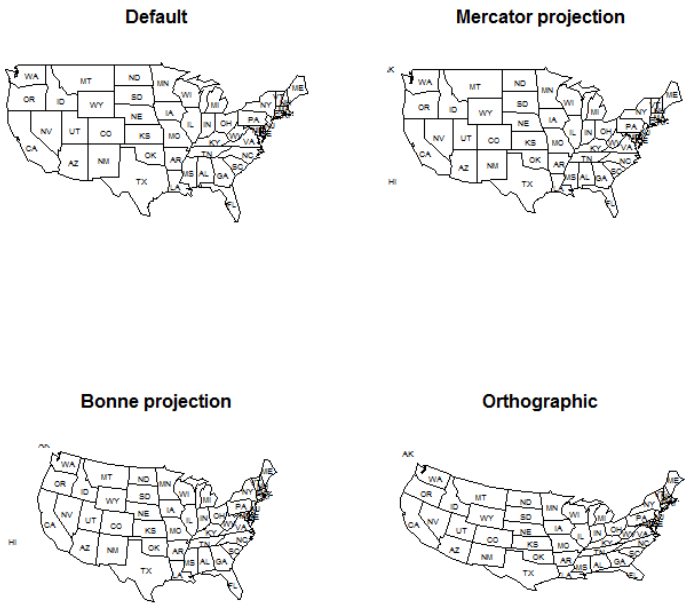

Figure 3: Some of the different projections available in the mapproj package in R.

and will rely on the default projection from the map package, which is a rectangular projection with the aspect ratio chosen so that longitude and latitude scales are equivalent at the center of the picture. This means that the scales of the map are different at different latitudes. When making maps of local areas, this scaling difference is not usually noticeable, but you would not want to use an R map for navigation!

The map package

The `map` package is a powerful package to create maps in R. It contains some databases of coastline data and state boundaries, and allows the user to create a map very easily. It works by defining the database you would like to plot (i.e., usa) and potentially a region or regions of that databse (i.e., washington). You can also use `xlim` and `ylim` to define a region to map, but those limits have to be within the database range or an error will occur.

```
#Figure 4
library(maps)       #basic mapping functions and some data

#Map of the world
par(mfrow=c(2,2))
map()
map('usa')  # national boundaries
map('state') #state boundaries
map('state', region = c('washington', 'oregon', 'california'))  # map of three states
map.axes()
```

Figure 4: Different maps and regions using the maps function in R.

The `mapdata` package contains additional databases, such as worldHires, nz, and rivers. The `mapproj` package contains various projections to plot your map with.

After making a map, you can easily add to it using low-level plotting functions such as `points`, `lines`, and `symbols`. However, when using a projection other than the default projection, you will need to convert from lat and lon to the coordinate system of that projection using `mapproject`. This is one reason that I use the default projection.

The data structure for a map object (of class map) is simply a list of items with `x` and `y` indicating polygons separated by NA, a range, and names of the regions (or polygons). Try typing `x <- map('state',region=`washington")" and examine what x consists of. Therefore it is easy to create your own map database by creating a list of x, y, range, and names, and then assigning the class of your object to map (i.e., `class(mymap)<- 'map'`).

The package `maptools` contains many functions to help expand your maps. For example, `readShapePoly` can read in polygon shapefiles, then plot them on a map. It is useful to look through the contents of this package as there are many interesting tools.

Using these four packages allows one to easily create maps that are suitable for publication and add data to these maps. This is not the only way to create maps and I will discuss the `PBSmapping` package next.

The PBSmapping package

I will only introduce the `PBSmapping` package here and hope that you will investigate it further as it is very powerful and can produce some beautiful maps. This is a package developed by the Pacific Biological Station in Nanaimo, Canada and is very useful for making maps of the NE Pacific, although it is not limited to that area. The package is currently not listed when choosing `Install Package(s)...` within the R console. It is available at `http://cran.open-source-solution.org/web/packages/PBSmapping/index.html` as a compressed file (e.g., zip) and can be downloaded and then installed into R 2.15 using
`Install package(s) from local zip files....`

The main function in the `PBSmapping` package is `plotMap`. It plots the data from a PBSmapping data type, which will be discussed later. One useful set of shoreline data for `PBSmapping` is `nepacLL` (for which there is also a `nepacLLhigh`). You first load the data using the `data` statement, then you can plot it.

```
library(PBSmapping) #powerful mapping functions developed by Pacific Biological Station
data(nepacLL)
data(nepacLLhigh)
windows(width=9,height=7)  #you have to experiment with dimensions to get axes labels, etc.
par(mfrow=c(1,2))
plotMap(nepacLL, xlim=c(-125, -121.9), ylim=c(46, 48.9),main="Low res (nepacLL)",
    col="green",bg="lightblue")
plotMap(nepacLLhigh, xlim=c(-125, -121.9), ylim=c(46, 48.9),main="High res (nepacLLhigh)",
    col="green",bg="lightblue")
```

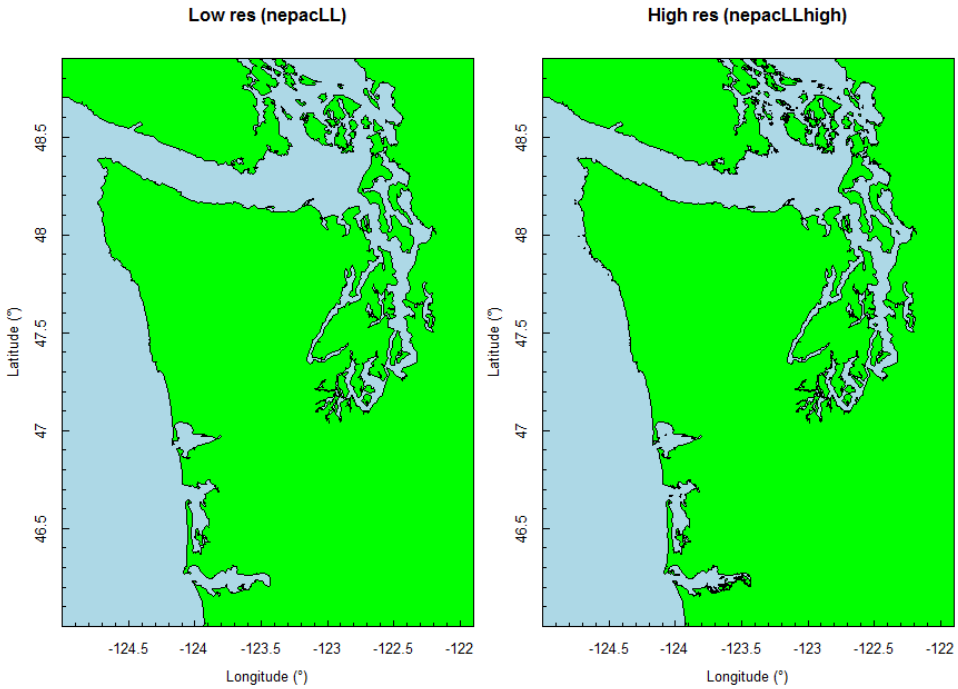

Figure 5: Different maps and regions using the maps function in R.

You can use low-level plotting functions to add points, lines, symbols, etc., as with the `maps` package, or `PBSmapping` has some built in functions for plotting. You can read in shapefiles using `importShapefile` and plot it with `addPolys` or `addLines`. Also have a look at `addBubbles`.

The data structure for `PBSmapping` is quite different than in the `maps` package. It consists of a matrix with the following columns:

PID: Primary ID

SID: Secondary ID

POS: Position of coordinate within a PID and SID

X: The horizontal coordinate (longitude)

Y: The vertical coordinate (latitude)

A `PBSmapping` dataset is of the class PolySet and has an additional attribute called projection, which I have only used "LL", standing for lat-lon. You can use `as.PolySet` to create a PolySet for `PBSmapping`.

There is a User Guide for `PBSmapping` that is easily found by searching on the internet and is very useful.

Other mapping packages

rworldmap: Maps of countries with attributes

mapplots: data visualization on maps

geo: N. Atlantic used by HAFRO for fisheries

sp: Spatial data

Imap: Point and click interface

ggmap: plot on top of Google maps

gmt: interface with GMT mapping software

plotGoogleMaps: data on Google maps

RgoogleMaps: Google map backgrounds

Advanced concepts

Creating your own depth contours from bathymetric data

Depth soundings are available online through sources like GeoDAS, and can be converting into contours for specific depths. The `makeTopography` function from the `PBSmapping` package will convert the xyz format downloaded from GeoDAS into a format suitable for the `contourLines` function. This process can take a long time depending on the amount of data downloaded.

```
#bathymetry data from GeoDAS or xyz format soundings
#xyz files, works best if you can get gridded data
#also see PBSmapping manual for free sources of bathymetric data
#these functions can take a long time, so I save the results to read in later
#plus the xyz file is very large.
xyz <- read.table("mapData\\westCoast.xyz",header=F,col.names=c("x","y","z"))
xyz$z <- -1*xyz$z
apply(xyz,2,range)
#reads in x,y,z data and make it useable by contours or contourLines functions
wcBathy <- makeTopography(xyz,digits=1)  #use smaller digits if data are sparse to make smoothe
save(wcBathy,file="mapData\\wcBathy.rda")

load("mapData\\wcBathy.rda")
plotMap(nepacLLhigh, xlim=c(-128, -121), ylim=c(45, 49),bg="lightblue",col="darkgreen")
xyzCL <- contourLines(wcBathy,levels=c(50,100,200))
xyzCP <- convCP(xyzCL)   #convert to poly set
wcPoly <- xyzCP$PolySet
attr(wcPoly,"projection") <- "LL"
addLines(wcPoly,col=c("darkblue","blue","darkcyan"),lty=1)
```

Shoreline data

There are also shoreline data available online that may be higher resolution and more up to date than what is included in the mapping packages. The `PBSmapping` user guide discusses the availability of these data and explains how to include them in your maps. The Global Self-consistent, Hierarchical, High-resolution Shoreline (GSHHS) database is a good source for data for the entire planet.

Compass Rose

This function plots a compass rose (north arrow) on the map at the specified location.

```
northarrow <- function(loc,size,bearing=0,cols,cex=1,...) {
    #From Tanimura, Kuroiwa, Mizota 2007. J. Statistical Software. V 19
    # checking arguments
    if(missing(loc)) stop("loc is missing")
    if(missing(size)) stop("size is missing")
    # default colors are white and black
    if(missing(cols)) cols <- rep(c("white","black"),8)
    # calculating coordinates of polygons
    radii <- rep(size/c(1,4,2,4),4)
    x <- radii[(0:15)+1]*cos((0:15)*pi/8+bearing)+loc[1]
    y <- radii[(0:15)+1]*sin((0:15)*pi/8+bearing)+loc[2]
    # drawing polygons
    for (i in 1:15) {
        x1 <- c(x[i],x[i+1],loc[1])
        y1 <- c(y[i],y[i+1],loc[2])
        polygon(x1,y1,col=cols[i])
    }
    # drawing the last polygon
    polygon(c(x[16],x[1],loc[1]),c(y[16],y[1],loc[2]),col=cols[16])
    # drawing letters
    b <- c("E","N","W","S")
    for (i in 0:3) text((size+par("cxy")[1])*cos(bearing+i*pi/2)+loc[1],
                        (size+par("cxy")[2])*sin(bearing+i*pi/2)+loc[2],b[i+1],cex=cex)
}

plotMap(nepacLLhigh, xlim=c(-128, -121), ylim=c(45, 49),bg="lightblue",col="darkgreen")
northarrow(c(-127,48),size=0.5)
```

1 Exercises

Easy Draw a map of New Zealand, including the Chatham Islands which are quite far to the east. Plot the following hot spots and label the "Spawning Box" with a name and arrow. (Hint: there is a high resolution map of New Zealand in the mapdata package).

```
locs <- data.frame(name=c("Challenger","Ritchie Banks","Spawning Box","Eastern Hills"),
                   x=c(168.5,178.4,183,185.7),
                   y=c(39,39.45,43.5,43.3)
)
```

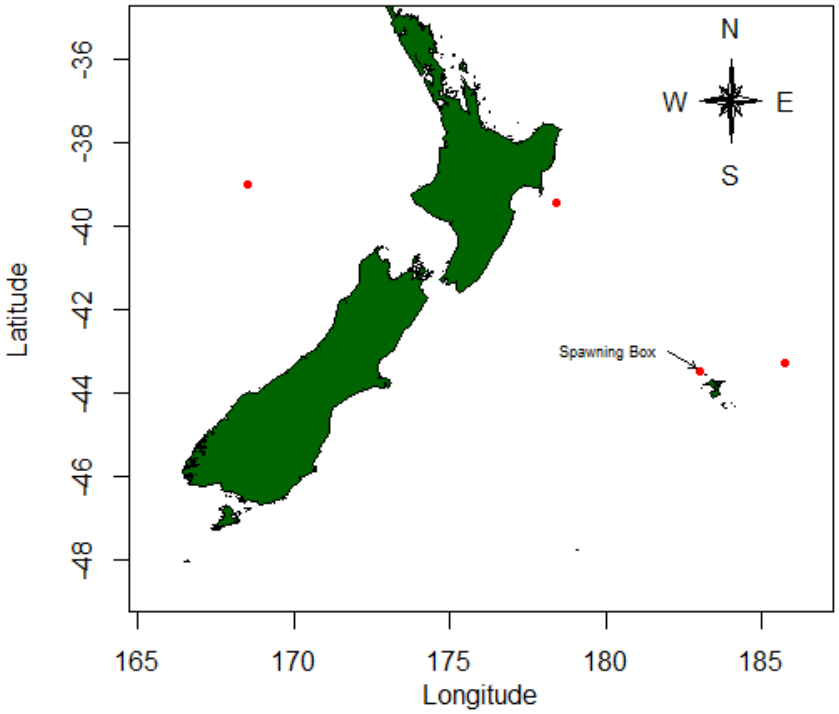

Figure 6: Map of New Zealand for the easy exercise.

Difficult Draw three maps of the USA with state boundaries. On two of these maps, plot the election results from 2008 and from 2012 (`ElectionResults2008.csv` and `ElectionResults2012.csv`) where red indicates that McCain or Romney won the majority, blue indicates that Obama won the majority, and green indicates where Other won the majority. On the third map, color the states won by Obama in 2008 and 2012 as blue, the states won by McCain in 2008 and by Romney in 2012 as red, and the states won by the opposite party in 2012 in a different color.

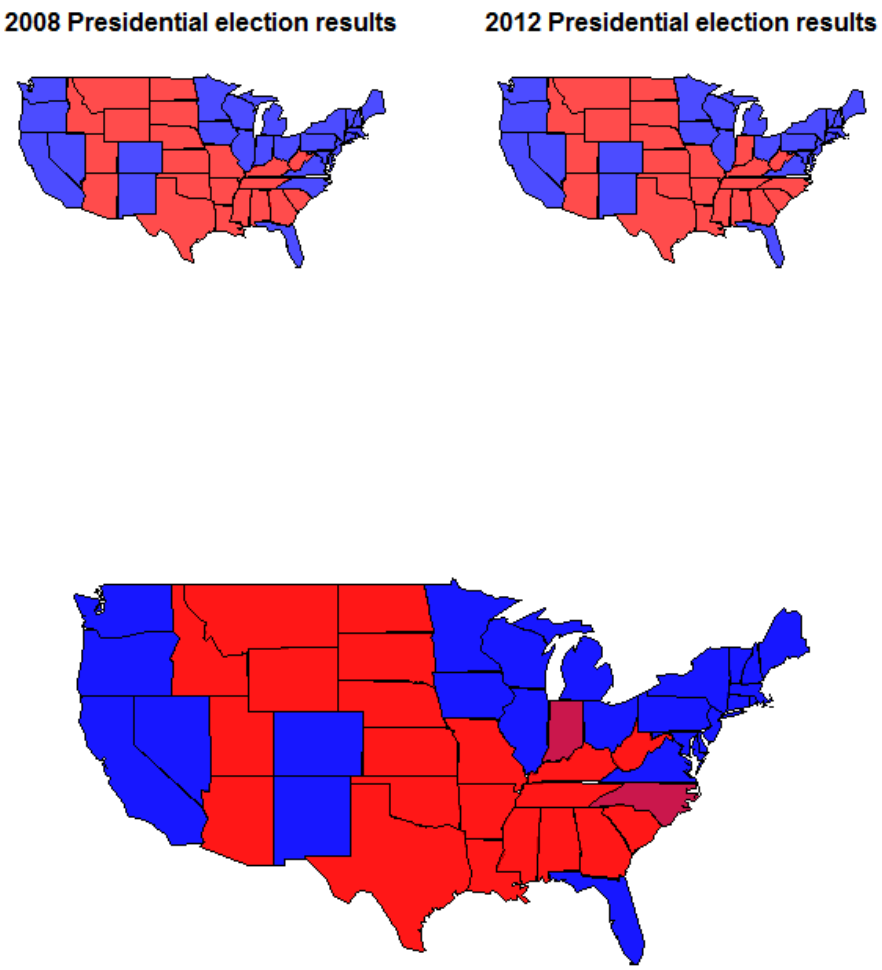

Figure 7: Map for the difficult exercise.

Advanced Draw a map of the USA with state boundaries and names. In the Pacific Ocean, place a small histogram (can be just `hist(rnorm(100))`) with an arrow pointing to California. Hint, use `split.screen`.

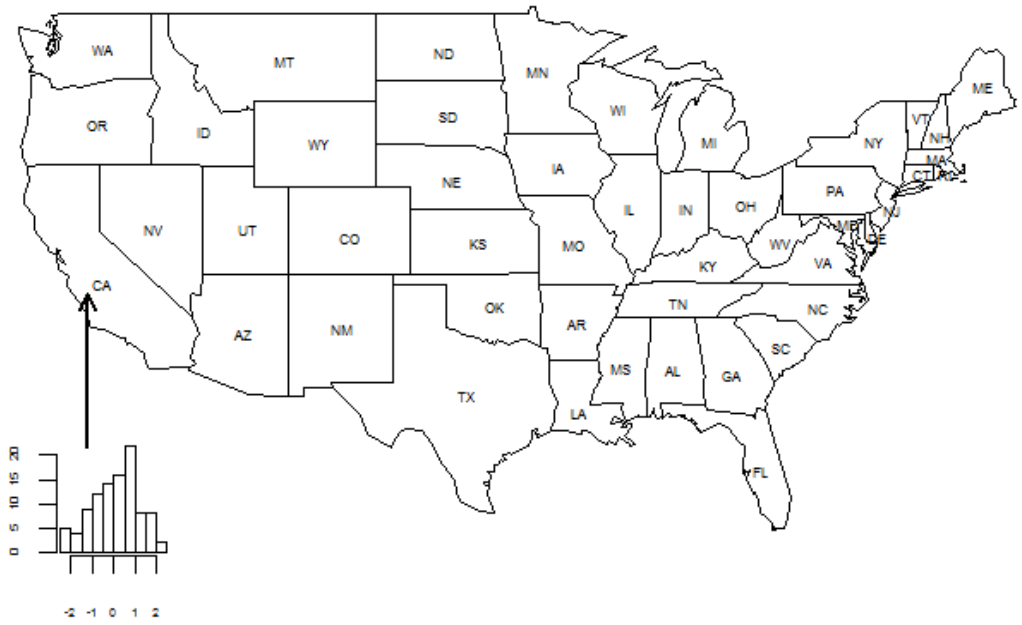

Figure 8: Map for the advanced exercise.