Mapping Great Circles Tutorial

Charles DiMaggio

February 21, 2012

I came across what may very well be one of my all time favorite uses of R graphics. It was by Paul Butler, and presents a visualization of pairs of Facebook relationships.

Figure 1: Facebook Connections. An R graphic by Paul Butler

I thought it would be a great way to represent transportation networks in New York City.

I tracked down a tutorial by Nathan Yau that presents the tools to come up with something like it. The motivation he presents is to draw "Great Circles" showing U.S. flights for major airlines. [1]

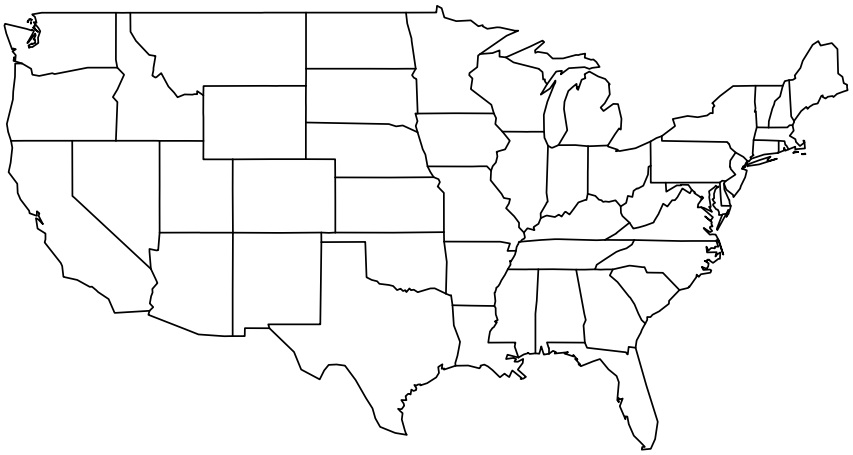
The two necessary packages are "maps" (for, logically enough, a map to draw) and "geosphere" which actually draws the arcs.

```
> library(maps)
> library(geosphere)
```

[1] The shortest distance between any two points on a sphere is an arc of a great circle, which is basically a circumferential line on the surface of the globe. Pilots use great circle routes whenever possible.

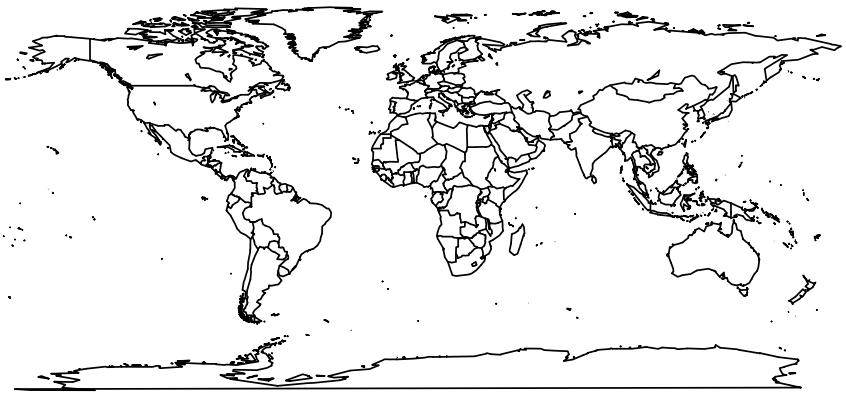
We begin by looking at a map of the United States.

```
> map("state")
```



Nice enough, but the projection leaves something to be desired, and it is missing Alaska and Hawaii. To get a better map, we will first map the world, then restrict to the area in which we are interested.

```
> map("world")
```



Now, we limit the map to a swath that includes the continental US, Alaska and Hawaii. We will also set some aesthetic values to set up the map so we can overlay some lines.

```
> xlim <- c(-171.738281, -56.601563)
> ylim <- c(12.039321, 71.856229)
> map("world", col = "#f2f2f2", fill = TRUE, bg = "white", lwd = 0.05,
+     xlim = xlim, ylim = ylim)
```

To draw an arc on the map, we first have to plot it. To do this, we use the *gcIntermediate()* function in "geosphere". The function takes as arguments the beginning and ending latitude and longitude (here the centroids for California and Maine), the number of points (n) we want plotted along the arc between those two points, and whether we want to include the beginning and ending points. The *lines()* function actually draws the line on the existing map.

```
> lat_ca <- 39.164141
> lon_ca <- -121.640625
> lat_me <- 45.213004
> lon_me <- -68.90625
> inter <- gcIntermediate(c(lon_ca, lat_ca), c(lon_me, lat_me),
+     n = 50, addStartEnd = TRUE)
> map("world", col = "#f2f2f2", fill = TRUE, bg = "white", lwd = 0.05,
+     xlim = xlim, ylim = ylim)
> lines(inter)
```

The "col" argument to *lines()* allows us to change the color of the line, which we do here for a line between California and Texas.

```
> lat_tx <- 29.954935
> lon_tx <- -98.701172
> inter2 <- gcIntermediate(c(lon_ca, lat_ca), c(lon_tx, lat_tx),
+     n = 50, addStartEnd = TRUE)
> map("world", col = "#f2f2f2", fill = TRUE, bg = "white", lwd = 0.05,
+     xlim = xlim, ylim = ylim)
> lines(inter)
> lines(inter2, col = "red")
```

Now that we can plot a line or two, we can move to plotting many lines. We begin by loading two data files. The flights data set was created by Nathan Yau from data he downloaded from the US DOT. It consists of flight counts between each airport in the US, categorized by airline. The airports file is airport latitude and longitude coordinates, obtained from the ASA.

```
> airports <- read.csv("/Users/charlie/Dropbox/greatCirclesMap/grtCrclTutorial/airport
+     header = TRUE)
> flights <- read.csv("/Users/charlie/Dropbox/greatCirclesMap/grtCrclTutorial/flights.
+     header = TRUE, as.is = TRUE)
```

In the following code, we first bring up the map. We subset the flight data by restricting to American Airlines. We loop over the airports data to get the beginning and ending airports for the restricted American Airlines data set. Then we use the *gcIntermediate()* function as we did above to create lines for the latitude and longitudes of the beginning and ending airports. Lastly, we add those lines to the plot.

```
> map("world", col = "#f2f2f2", fill = TRUE, bg = "white", lwd = 0.05,
+     xlim = xlim, ylim = ylim)
```

```
> fsub <- flights[flights$airline == "AA", ]
> for (j in 1:length(fsub$airline)) {
+     air1 <- airports[airports$iata == fsub[j, ]$airport1, ]
+     air2 <- airports[airports$iata == fsub[j, ]$airport2, ]
+     inter <- gcIntermediate(c(air1[1, ]$long, air1[1, ]$lat),
+         c(air2[1, ]$long, air2[1, ]$lat), n = 100, addStartEnd = TRUE)
+     lines(inter, col = "black", lwd = 0.8)
+ }
```

We can scale the color of the lines to reflect the number of flights using the *colorRampPalette()* to create a function [2] that will interpolate a spectrum of colors based on a base color (here black). You pass a number (here 100) to the function you create with *colorRampPalette()* (here called "pal") to define the scale. The scale itself is created (and called "colindex" in the code below) based on the number of flights on a route as a proportion of all the flights on a route (the variable "maxcnt" created in line 5 of the code below).

```
> pal <- colorRampPalette(c("#f2f2f2", "black"))
> colors <- pal(100)
```

[2] That's right. It's a function that creates a function.

```
> map("world", col = "#f2f2f2", fill = TRUE, bg = "white", lwd = 0.05,
+     xlim = xlim, ylim = ylim)
> fsub <- flights[flights$airline == "AA", ]
> maxcnt <- max(fsub$cnt)
> for (j in 1:length(fsub$airline)) {
+     air1 <- airports[airports$iata == fsub[j, ]$airport1, ]
+     air2 <- airports[airports$iata == fsub[j, ]$airport2, ]
+     inter <- gcIntermediate(c(air1[1, ]$long, air1[1, ]$lat),
+         c(air2[1, ]$long, air2[1, ]$lat), n = 100, addStartEnd = TRUE)
+     colindex <- round((fsub[j, ]$cnt/maxcnt) * length(colors))
+     lines(inter, col = colors[colindex], lwd = 0.8)
+ }
```

The problem here is that the less frequent flights (and hence lighter-colored lines) obscure the more frequent flights because the lines are being drawn in the order in which they occur in the data. We use the *order()* function in the 6th line of the code below, to layer the darker (higher count) flights over the lighter ones. You can change the color scheme if you like. Here, we've used two colors, so the ramp goes from

```
> pal <- colorRampPalette(c("#f2f2f2", "black"))
> colors <- pal(100)
> map("world", col = "#f2f2f2", fill = TRUE, bg = "white", lwd = 0.05,
+     xlim = xlim, ylim = ylim)
> fsub <- flights[flights$airline == "AA", ]
> fsub <- fsub[order(fsub$cnt), ]
> maxcnt <- max(fsub$cnt)
> for (j in 1:length(fsub$airline)) {
+     air1 <- airports[airports$iata == fsub[j, ]$airport1, ]
+     air2 <- airports[airports$iata == fsub[j, ]$airport2, ]
+     inter <- gcIntermediate(c(air1[1, ]$long, air1[1, ]$lat),
+         c(air2[1, ]$long, air2[1, ]$lat), n = 100, addStartEnd = TRUE)
+     colindex <- round((fsub[j, ]$cnt/maxcnt) * length(colors))
+     lines(inter, col = colors[colindex], lwd = 0.8)
+ }
```

Here, we fiddle around a little bit more with the colors. You can get some hex color numbers here

```
> pal <- colorRampPalette(c("#333333", "white", "#1292db"))
> colors <- pal(100)
> map("world", col = "#f2f2f2", fill = FALSE, bg = "#191980", lwd = 0.1,
+     xlim = xlim, ylim = ylim)
> fsub <- flights[flights$airline == "AA", ]
> fsub <- fsub[order(fsub$cnt), ]
> maxcnt <- max(fsub$cnt)
> for (j in 1:length(fsub$airline)) {
+     air1 <- airports[airports$iata == fsub[j, ]$airport1, ]
+     air2 <- airports[airports$iata == fsub[j, ]$airport2, ]
+     inter <- gcIntermediate(c(air1[1, ]$long, air1[1, ]$lat),
+         c(air2[1, ]$long, air2[1, ]$lat), n = 100, addStartEnd = TRUE)
+     colindex <- round((fsub[j, ]$cnt/maxcnt) * length(colors))
+     lines(inter, col = colors[colindex], lwd = 0.8)
+ }
```