Workshop 1: Plotting Vector Spatial Data

R-Wizardry Final Project

April 13, 2016

Jenn Retzlaff – University of Calgary

Introduction

If you are looking to manipulate and plot vector data onto google maps, without the use of ArcGIS or other spatial analysis software, R offers a powerful toolkit for map-making and spatial analysis. Vector data in particular is useful for mapping data onto maps nicely as it scales well and can be represented at many different resolutions. Often land cover data is procured in vector formats, allowing users to manipulate and scale visuals in a way that is appropriate for their projects.

Vector data is available in three formats, point data (point coordinates), line data (sequences of coordinates), and polygon data (sequences of coordinates connected as a loop). This tutorial will deal with basic map-making tool using vector data in the format of point data, functions to overlay data once you have an appropriate map generated. R is a powerful tool to quickly input many data points, eliminating repetitive analytical tasks and automating multi-step extension.

After this tutorial, you should be able to create or extract vector data, produce maps from Google maps and display basic locality and abundance data associated with longitudes and latitudes. Images of output are provided throughout the tutorial so that you may follow along.

The dataset we will be using deals with spatial data for Bumble bees in southern Alberta and is titled, “Dataset”, see attached.

Workspace

Begin the usual way by clearing your directory, setting your working directory and reading in the dataset that we will be using in this tutorial.

```
rm(list=ls(all=T)) # clear any previous work
setwd("C:") #set working directory
data <- read.csv("Dataset.csv",header=T,stringsAsFactors = FALSE) #read
in dataset from working directory
```

Loading packages

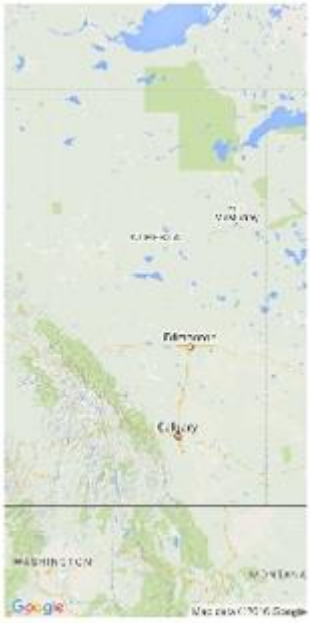
The sp package is the official spatial analysis package in R, however, there are many useful packages that include functions that allow extraction and manipulation of maps from the internet, some of which I will discuss. There are many packages that might be useful to you in your exploration of map creation and data plotting within R, for the purposes of this tutorial, packages to be installed include:

```
install.packages( c("sp", "raster", "rastervis", "maptools",
                    "rgeos", "dismo", "rgdal", "XML", "ggmap", "GISTools"))
library(sp)  # classes for spatial data
library(raster)
library(rastervis)
library(maptools) #allows us to work with shapefiles and other formats
library(rgeos)
library(dismo)
library(rgdal)
library(XML)
library(ggmap)
library(GISTools)
```

Map extraction

1.1 Basic Google map retrieval using gmap function in dismo package and plot function to display the map.

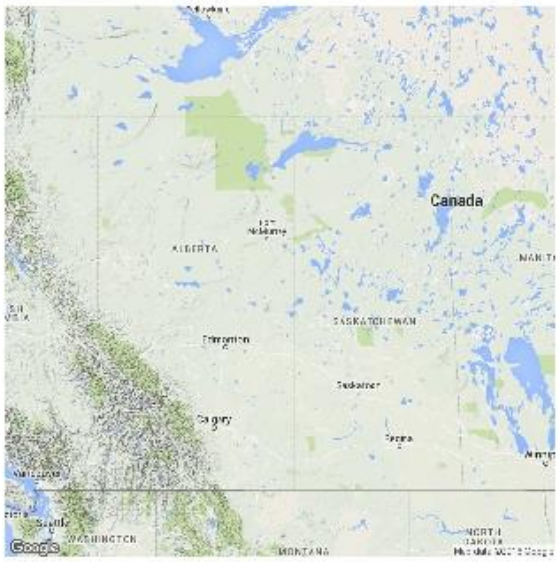
```
##EXTRACTING A MAP
mymap <- gmap("Alberta", type='roadmap', zoom=5, scale=2)  # choose whatever country/region
plot(mymap, inter=T)
```



There are many arguments that you may choose to include, depending on what you would like your map to look like. You may call for a country, province, or state (known as “textual” localities), or you can use an extent object defined by longitude/latitude coordinates (see below). Options for type include roadmap, satellite, hybrid, or terrain maps. Zoom determines the size of area you will be shown, where 0 would be the whole world and 21 would be a very small section centered on the center of your extent or locality. Scale (1 or 2 only) determines the resolution of your map where 2 is the higher resolution option.

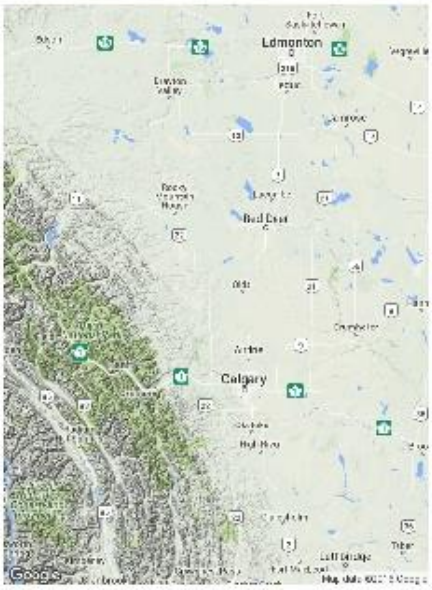
1.2 You may also choose to use the extent function to create a map by defining the boundaries of the map using latitudes and longitudes:

```
e = extent( -100.9531 , -120.3897 , 50.36 , 60.61956 )
r = gmap(e)
plot(r)
```



1.3 Given a map that you have extracted, you may choose to work with a smaller area within the map, and you can do this by using the drawExtent function.

```
select.area <- drawExtent()
# now click 2 times on the map to select your region
mymap <- gmap(select.area)
plot(mymap)
```



1.4 Before moving on to plotting data onto maps, it is important to note that maps can be saved when generated by including the filename argument which will save the map to your working directory.

Example:

```
mymap <- gmap("Alberta", type='roadmap', zoom=5, scale=2, filename = "Alberta.map")
plot(mymap, inter=T)
```

Vector data creation and mapping it onto maps

There are many ways to map vector data, depending on whether you want to create your own coordinate system, input selections of data, or utilize a pre-existing data base of coordinates.

2.1 Creation of vector data. You may create individual objects for longitude and latitude and use the cbind function to create an object to be plotted containing the entire coordinate or coordinate set.

```
##CREATING COORDINATES
x = runif(30)*10 + 40
y = runif(30)*10 - 20
xy = cbind(x, y)
```

2.2 Another option is to use data from our file “Dataset”, as we can read in columns of data containing longitude and latitude and use cbind to bind the data together to be plotted.

```
x = (data$lon)
y = (data$lat)
xy = cbind(x, y)
```

2.3 Once you have plotted your map based on your new set of coordinates, it is simple to add the locations points onto your map using the points function.

Example using the locality information extracted in 2.2 that will now define the extent of the map.

```
g = gmap(xy, type='hybrid', lonlat=T)
plot(g, inter=TRUE)
points(xy , col='red', pch=20)
```

2.4 One useful application of the gmap function is the ability to manipulate the plotted points to represent different aspects of the dataset.

In our case, we can manipulate the data set to reflect abundance, but first we need to manipulate the dataset to calculate abundance data. Prior to this, each unique locality has been plotted on the map, but ignored the number or count of data points at each locality. This can be done by creating a new data set where the abundance of individuals in the population (number of BBIDs) that are associated with a location (locality defined by latitude and longitude) can be summarized.

```
locs=select(data,BBID,lat,lon,locality)%>% #Gets BBID,lat,lon,locality from "data"
  group_by(locality)%>% #Groups by locality
  summarize(totalbees=n(),lat=lat[1],lon=lon[1]) #for each locality, get a value
to represent abundance
```

This should produce a new data frame with a column specifying abundance data (totalbees) that looks like this:

	locality	totalbees	lat	lon
1	0.2 km from Hartell	8	50.6010	-114.2356
2	0.2 km from Mossleigh	2	50.7194	-113.3289
3	1.2 km from Herronton	8	50.6305	-113.4519
4	1.3 km from Eltham	40	50.5720	-113.5471
5	1.6 km from Farrow	16	50.6888	-113.4288
6	10.5 km from Gladys	154	50.7766	-113.8366
7	10.5 km from Mossleigh	21	50.6304	-113.2782
8	11.2 km from Mossleigh	8	50.7347	-113.1672
9	11.2 km from Okotoks	21	50.8050	-113.8709
10	112 St W, De Winton	30	50.7729	-114.1633
11	12.6 km from Okotoks	4	50.7986	-114.1404

Once we have this new data frame, we can extract and manipulate the lat/lon data as before and plot, again using gmap function. The cex argument is used to implement the newly created abundance data, as it will use the numerical argument given to magnify the point relative to the default size=1. This will produce points which reflect the abundance at each locality by using points that are scaled according to abundance.

```
x = (locs$lon)
y = (locs$lat)
xy = cbind(x, y)
g = gmap(xy, type='hybrid', lonlat=T)
plot(g)
points(xy , col='red', pch=1, cex= locs$totalbees*0.03)
```

Additions to your map

3.1 Adding a scale bar to your map

```
scalebar(20, label=c(0,10,20), below="Kms", type="bar", divs=2)
```

This scale bar is customizable, and can be adjusted to cover different distances (Km), you may choose to place the bar onto your map by specifying coordinates, it can be a “line” or “bar”, and you can divide the scale bar as many times as you wish. The label argument allows you to specify beginning, midpoint and end points for the scale bar, and the below function allows you to add text such as “Km” or “Miles” to define the units used.

3.2 Add a north arrow to you figure

```
north.arrow(xb=-114.4400, yb=50.6000, len=0.01, lab="N", col='white')
```

The arrow is drawn using polygon data and is described by the arguments specified to determine location, size and colour.

The arguments xb and yb specify the location for the arrow base (using map units) and len determines the length of the arrow base. In 2.3, I included the argument lonlat=T, this was important for this function as it allows us to use latitude and longitudes to determine the north arrow location on the map. The argument col determines the colour of the arrow, and tcol specifies the colour of the label text.

Output

Your final graphics should look like this, a map, with all localities shown with sizing representing relative abundance at each site. Scale bar in kilometers as well as north arrow are depicted