RGeoSpatial

BFI-EPIC Orientation 2020

Santiago Lacouture

July 23rd

Requirements

- Some basic knowledge of R
- `sp`: working with vector data,
- `rgdal`: importing and exporting vector data from other programs,
- `rgeos`: manipulating spatial objects

`sp` can be directly installed using `install.packages` on both Windows and Mac. It only works on `rgdal` and `rgeos` for Windows.

To install `rgdal` on Mac:

1. Download GDAL complete
2. Doubleclick and install the `.dmg` file
3. Download the rgdal package from CRAN.
4. Place the downloaded `rgdal_1.0-4.tgz` in a folder you remember and run `install.packages("DIR", repos=NULL)`

Then, to install `rgeos` you should be ready to go by typing

```
install.packages("rgeos", type = "source", configure.args =
    "--with-geos-config=/Library/Frameworks/GEOS.framework/Versions/Current/
    unix/bin/geos-config")
```

This session

1. Background on Projections, CRS and GCS
2. Intro to Spatial Vector Data in R
3. Vector Data manipulation and visualization

This session

1. Background on Projections, CRS and GCS
2. Intro to Spatial Vector Data in R
3. Vector Data manipulation and visualization

How Computers Relate x-y Coordinates to Real Locations on Earth

- What on earth do the coordinates (x,y) mean?
- This requires two things:
  - a model of the real, three dimensional globe, and
  - a specification for how to convert locations on a spherical object into a 2-dimensional representation.

BLUF

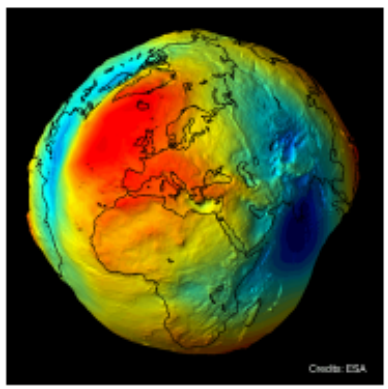
It *usually* doesn't matter how you do it, but you **must** be consistent.

Model of the Globe

- When two sources provide the same latitude and longitude, those may not always correspond to the same real world location.

Smooth sphere

Sphere with elevations

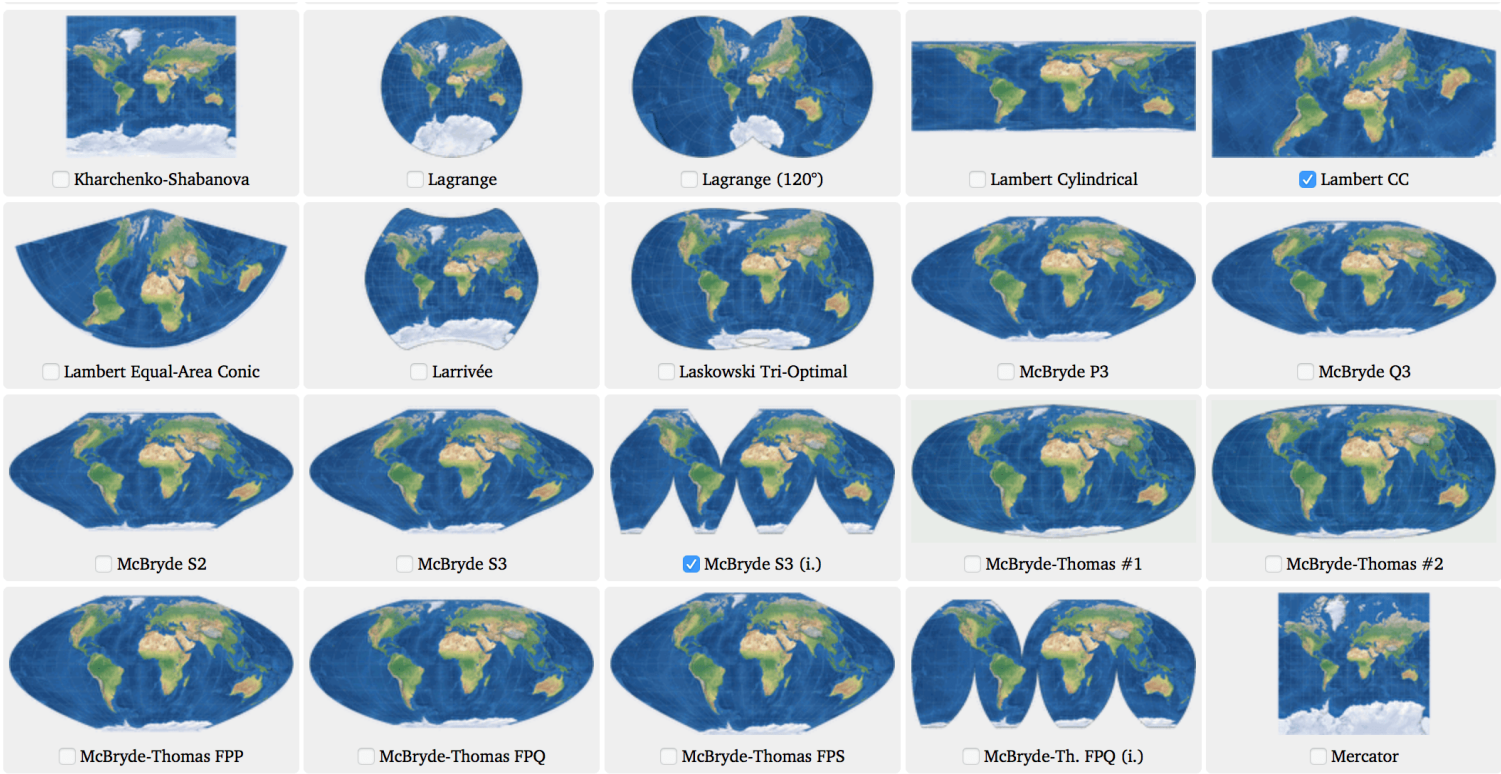


Satellite-based model of earth[1]

Taken from Nick Eubank

Flattening the globe

- Earth is round, but our computer screens are not.



- There is no such thing as the "correct" way – different approaches are just different distortions.

Terminology

- **Projection:** A Flattening Function, but in some programs (like ArcGIS), the term Projected Coordinate System is used to refer to a bundle that includes both a Globe Model and a Flattening Function.
- **Geographic Coordinate System (GCS):** The ArcGIS term for just a Globe Model.
- **Coordinate Reference System (CRS):** term used by GIS packages in R to define both a Globe Model and a Flattening Function.

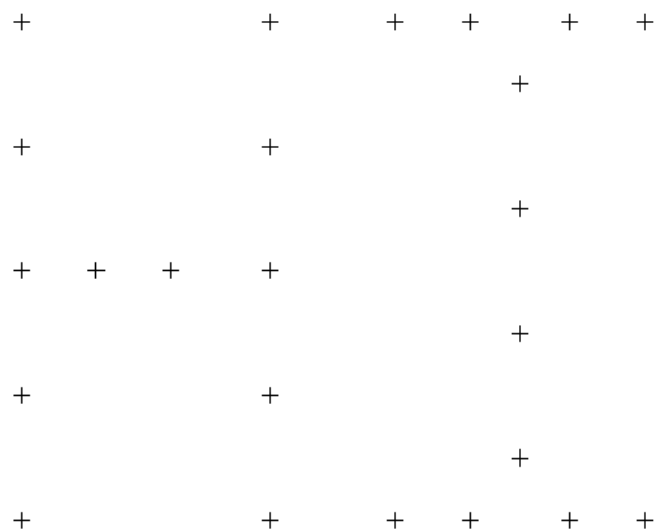
This session

1. Background on Projections, CRS and GCS
2. Intro to Spatial Vector Data in R
3. Vector Data manipulation and visualization

SpatialPoints (1/2)

- A point is a pair of x-y coordinates
- SpatialPoints is a collection of points

```
my.points <- rbind(c(1,1), c(1,1.5),
    c(1,2), c(1,2.5), c(1,3), c(2,1),
    c(2,1.5), c(2,2), c(2,2.5),
    c(2,3), c(1.3,2), c(1.6,2),
    c(2.5,1), c(2.8,1), c(3.2,1),
    c(3.5,1), c(3,1.25), c(3,1.75),
    c(3,2.25), c(3,2.75), c(2.5,3),
    c(2.8,3), c(3.2,3), c(3.5,3))
my.first.points <-
    SpatialPoints(my.points)
plot(my.first.points)
```



SpatialPoints (2/2)

- SpatialPoints has the ability to keep track of how the coordinates relate to places: CRS
- This is stored in an object called `proj4string`
- CRS objects can be created by passing the `CRS()` function the code associated with a known projection.
- You can find the codes for most commonly used projections here.

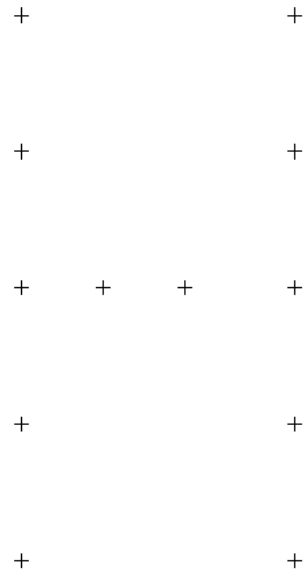
```
summary(my.first.points)
is.projected(my.first.points)
```

```
crs.geo <- CRS("+init=EPSG:32633")
proj4string(my.first.points) <-
    crs.geo
is.projected(my.first.points)
```

SpatialPointsDataFrame

- SpatialPointsDataFrame is a SpatialPoints with attributes added in a data.frame (saved at `@data`).
- In a simple combination, points will be merged based on the order of observations.

```
df <- data.frame(letter = c(rep("H",12),
    rep("I",12)))
my.first.spdf <- SpatialPointsDataFrame(
    my.first.points, df)
summary(my.first.spdf)
```



- With attributes we can manage spatial data like a data.frame

```
my.first.spdf[1:2, ]
plot(my.first.spdf[which(
    my.first.spdf$letter == "H"), ])
```

SpatialPolygons

- A `Polygon` is a single geometric shape (e.g. a square, rectangle, etc.)
- A `Polygons` consists of one or more `Polygon` objects that combine to form a unit of analysis.
- A SpatialPolygons is a collection of `Polygons` objects: R analogue of a `shapefile` or `layer`.

```
house.building <- Polygon(rbind(c(3,
    1), c(4, 1), c(4, 0), c(3, 0)))
house.roof <- Polygon(rbind(c(3, 1),
    c(3.5, 2), c(4, 1)))
house.door <- Polygon(rbind(c(3.25,
    0.75), c(3.75, 0.75), c(3.75, 0),
    c(3.25,0)), hole = TRUE)
```

```
house <-
    Polygons(list(house.building,
    house.roof, house.door), "house")
```

```
frame <- SpatialPolygons(list(house,
    tree))
```

SpatialData as Vectors

- Coordinate system: same idea.
- SpatialPolygonsDataFrame: just as SpatialPointsDataFrame
  - When you first associate a `data.frame` with a SpatialPolygons object, R will line up rows and polygons by matching `Polygons` object names with the `data.frame` `row.names`
  - After the initial association, this relationship is **no longer** based on `row.names`! Do not re-order the `data.frame` manually.
- SpatialLines: Just like SpatialPolygons

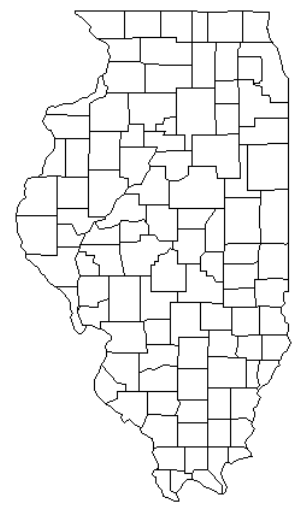
This session

1. Background on Projections, CRS and GCS
2. Intro to Spatial Vector Data in R
3. Vector Data manipulation and visualization

Importing and Exporting Spatial Data

- We can read in and write out spatial data using: `readOGR()` and `writeOGR()`
- `readOGR()` expects at least 2 arguments:
  1. Data Source Name (`dsn`): the path to the folder that contains the files,
  2. Layer Name (`layer`): the file name without extension.
- Example: Illinois county border.
  1. Download the ESRI shapefile of county borders in the US (try the "_20m" version).
  2. Unzip into a folder (let's call it "US counties"),
  3. Read it and subset based on attributes of data.

```
UScounties <- readOGR(dsn = "US counties",
    layer = "cb_2017_us_county_20m") # Read
    all counties
head(UScounties@data) ## Explore associated
    data
ILcounties <-
    UScounties[UScounties@data$STATEFP ==
    "17", ] # Restrict based on attribute
plot(ILcounties)
```



Spatial + Non-Spatial Join

- Tabular data + Spatial* object: direct association
- We use `merge` as with data.frame
- **NEVER** merge the `data.frame` associated with you Spatial* object directly.

Example: County COVID-19 cases in IL.

1. Download the data from USAfacts.
2. Read it as `data.frame` in R,
3. Merge it with the shapefile of US counties

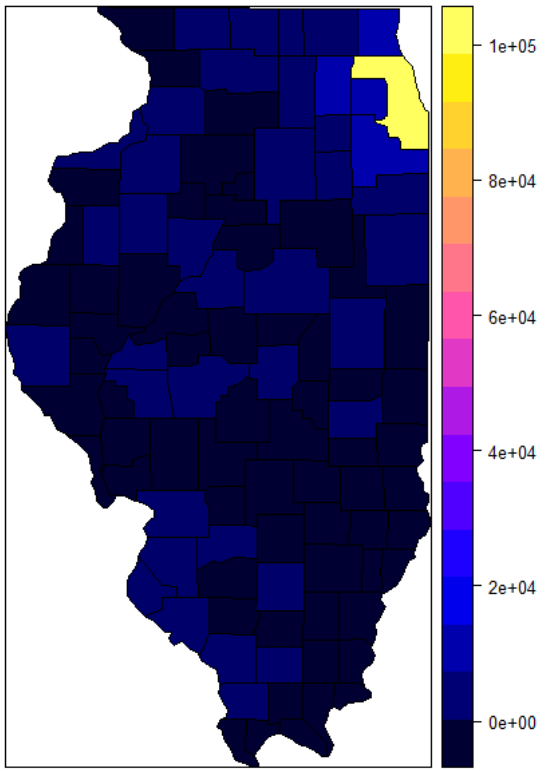
```
covid.cases <-
    read.csv("covid_confirmed_usafacts.csv")
UScounties@data$countyFIPS <- as.integer(
    as.character(
    UScounties@data$STATEFP))*1000
    +as.integer( as.character(
    UScounties@data$COUNTYFP))   # We have
    to make sure the key variable is the
    same in both data sets
UScounties <- merge(UScounties, covid.cases,
    by = "countyFIPS")
```

Basic Visualization of Information on Maps (1/2)

- The easiest way to plot maps with information is to use spplot
- Say we are interested again on IL:

```
ILcounties <-
    UScounties[UScounties@data$STATEFP ==
    "17", ]
spplot(ILcounties, "X7.19.20")
```

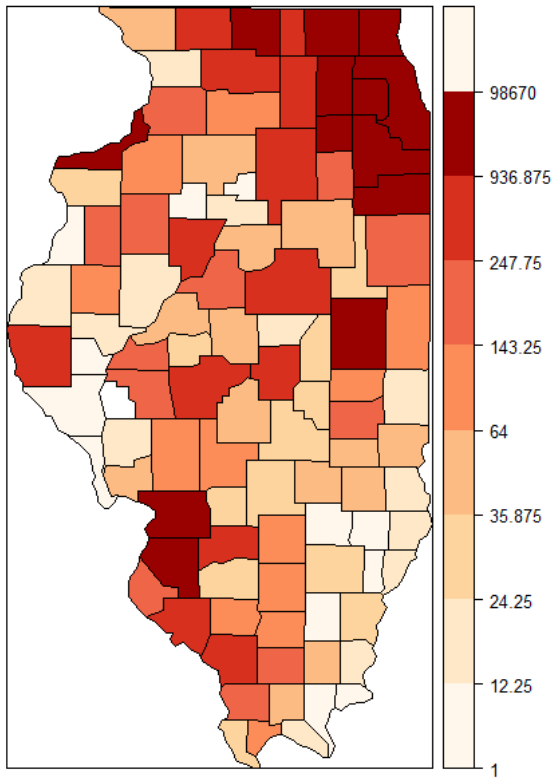
- The function will automatically create cuts and will use standard color set.



Basic Visualization of Information on Maps (2/2)

- We can modify the standard behavior of spplot:

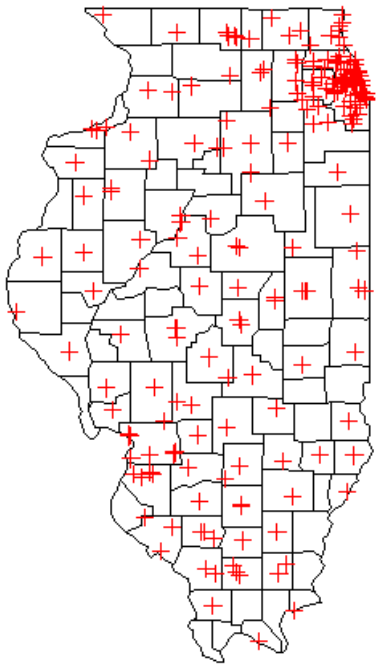
```
library(RColorBrewer) # Customize colors
my.palette <- brewer.pal(n = 9, name =
    "OrRd") # Say we want 8 categories
breaks.qt <- quantile(ILcounties$X7.19.20,
    probs = seq(0,1,0.125)) # create cuts of
    the variable
ILcounties@data$cut <-
    cut(ILcounties@data$X7.19.20, breaks =
    c(breaks.qt, Inf)) # create factor
    variable based on these cuts
spplot(ILcounties, "cut", col.regions =
    my.palette, colorkey = list(labels =
    list(at = seq(0.5, length(breaks.qt)
    -0.5), labels = breaks.qt)))
```



You can also plot maps using ggplot2. You have to get familiar with the sf library (more about this at the end). Check out more on this here

Spatial + Spatial (1/2)

- We are going to exemplify this with another shapefile: Hospitals in the US.
  - Download the shapefile and unzip it into a folder (we call it "Hospitals US")
  - Open it! Is a SpatialPointsDataFrame
- You have to make sure they have the same CRS:
  - You can retrieve the CRS from an existing Spatial* object with the `proj4string()` command
  - Then you use the `CRS()` function to read it as a CRS object
  - Finally you use `spTransform()` to reproject the other layer



```
hospitalsUS <- readOGR(dsn = "Hospitals US",
    layer = "Hospitals")
# Re-projection
common.crs <- CRS(proj4string(UScounties))
hospitalsUS.reprojected <-
    spTransform(hospitalsUS, common.crs)
```

- We can visualize this: we plot the 2 layers ussing `plot` and its option `add = T`

```
plot(ILcounties)
plot(hospitalsUS.reprojected[
    hospitalsUS.reprojected@data$STATE ==
    "IL", ], col = "red", add = T)
```

Spatial + Spatial (2/2)

- The primary tool for doing spatial joins is the `over` command: "For each item of first position (the SOURCE), `over` returns information about items of second argument (TARGET) that intersect"
- For example, we can extract the information we have about the counties for each hospital.
- We can then recombine this info into the Spatial object using standard vector operations into the `@data` attribute.

```
HospitalsIL <- hospitalsUS.reprojected[hospitalsUS.reprojected@data$STATE ==
    "IL", ]
ILHospitalsCounties <- over(HospitalsIL, ILcounties)
HospitalsIL@data[, colnames(ILHospitalsCounties)] <- ILHospitalsCounties
```

- By default, when there are multiple TARGET observations that intersect a SOURCE observation, `over` will just return the first one. There are two ways to address this.
  - `returnList = T` and `sapply`
  - `fn` option

Overview of Geometric Manipulations

`over` is just an example of how to manage and join Spatial objects in R. In general, you can use `rgeos` set of tools (from `sp` package). There are many other functions to:

- Calculate values
  - **gArea**
  - **gLength**
  - **gDistance**
- Create New Objects
  - **gBuffer**
  - **gCentroid**
  - **gUnion**, **gIntersection**, **gDifference**
  - **gSimplify**
- Testing Geometric Relationships
  - **gIntersects**
  - **gContains**

Key `rgeos` Concepts

- Units: `rgeos` just does geometry: it will take the `+units` from the projection of the object.
- `byid`: option of most commands:
  - = `T` each observation is handled separately (each),
  - = `F` object is treated as one big geometry (any).
- `id`: organization of the output of the functions. This is usually the rowname for a given observation in the operation.

Brief discussion about `sf`

- `sf`: has much the same functionality as `sp`, but promises to be much more convenient and flexible.
- `sf` works with a language independent standard for structuring spatial data termed 'simple features':
  - Intuitive data structures
  - Intuitive operations
  - Spatial indexing
  - BUT a lot of packages that depend on spatial classes still rely on `sp`.

Try it at home

Try to play with the following shapefile: Roads (primary and secondary) in Illinois. Download here

- Start plotting it against the county boundaries of IL
- Tray to answer the following Qs:
  1. How many different roads are located in each county?
  2. What is the largest number of counties the same road is partially located at?
  3. How many Kms of roads are located in each county?
  4. Use your imagination!

If you are interested in learning more about GIS in R, I'd recommend you to visit this repository of helpful resources.