GIS in R Command Cheat Sheet
Vector Data

Nick Eubank

October 1, 2015

Creating Spatial Objects From Scratch

Creating Points:

Points: `SpatialPoints([matrix of coordinates])`

- Note: if latitude and longitude coordinates, must be ordered longitude (x-coordinate), latitude (y-coordinate)

Points with DF: `SpatialPointsDataFrame([Spatial Points Obj], [DataFrame])`

Creating Lines:

Line (single geometric line): `Line([matrix of coordinates of vertices])`
Lines (single "observations" potentially consisting of several basic lines, like a river):
`Lines([list of Line Objs], [name for Lines objs])`
SpatialLines (collection of "observations", like shapefile):
`SpatialLines([list of Lines Objs])`
Spatial Lines with DF: `SpatialLinesDataFrame([SpatialLines Obj, DataFrame])`

Polygons:

Polygon (one geometric shape defined by a single enclosing line):
`Polygon([matrix of coordinates of vertices])`
Polygons (single "observations" potentially consisting of several basic shapes):
`Polygons([list of Polygon Objs], [name for Polygons objs])`
SpatialPolygons (collection of "observations", like shapefile):
`SpatialPolygons([list of Polygons Objs])`
Spatial Polygons with DF: `SpatialPolygonsDataFrame([SpatialPolygons Obj, DataFrame])`

- During creation, will link rows to polygons by matching DataFrame `row.names` to names of `Polygons`.
- After initial alignment based on names, relationship is based on order of rows.

Importing and Exporting

GPS Coordinates in Table:

1. Use `read.csv()` to import DataFrame with lat long coordinates.
2. `coordinates([DataFrame]) <- c([name of column with long],[name of column with lat])`
   - Note reverse ordering: longitude (x-coordinate), then latitude (y-coordinate).

Read Vector-Based Files:
`data <- readOGR(dsn=[path to FOLDER holding data], layer=[name of shapefile in folder])`

- Note: do not include extension (like `.shp`) in `layer` argument

Export Vector-Based Files:
`writeOGR([Spatial obj], dsn=[path to folder],layer=[name, no suffix], driver=[format])`

- Shapefile: `driver="ESRI Shapefile"`.
  Full format list

Interrogating Spatial Objects

Summaries:
Quick summary: `summary([Spatial obj])`
Longer summary of contents: `str([Spatial obj])`
Full list of contents: `attributes([Spatial obj])`
Check if projected: `is.projected([Spatial obj])`

Extract Attributes:
Bounding Box: `bbox([Spatial obj])` **or** `[Spatial obj]@bbox`
Get full projection info: `proj4string([Spatial obj])` **or** `[Spatial obj]@proj4string`
Get associated coordinates: `coordinates([Spatial obj])` **or** `[Spatial obj]@coordinates`
Get associated data: `[Spatial obj]@data`

Projections

Assigning projection by EPSG code: `proj4string([Spatial obj]) <-CRS("+init=EPSG:4326")`
Get projection from Spatial obj: `proj4string([Spatial obj])`
Re-project:

```
newProjection <- CRS("projection string goes here")
spTransform([Spatial object],newProjection)
```

Codes: www.spatialreference.org

Merging Data

Spatial* + Non-Spatial Table:
`library(plyr)` `mySpatial@data <- join(mySpatial@data, table, by = "cols", type="left")`

- **Never** use `merge`; use `join` from `plyr`
- Keep spatial data table in first position

Spatial* + Spatial*: `over([Spatial obj A],[Spatial obj B], fn=[NULL, or function if computing])`

- ENSURE PROJECTIONS MATCH FIRST!
- If B does not have data.frame, returns index of item in B that intersect with A.
- If B has DataFrame, returns observation for items in B that intersect with A.
- If multiple things in B intersect, returns FIRST ONLY. Pass `fn` to aggregate, or `returnList=TRUE` to get all intersects.
- Can't join two SpatialPolygons layers.

SpatialPolygons + SpatialPolygons: `gIntersects(SpatialObject1, SpatialObject2, byid=TRUE)`
Subset Intersecting Observations (like Polygons):

1. `select <- gIntersects(SpatialObject1,SpatialObject2,byid=TRUE)`
2. `selected.SpatialObjects <- SpatialObject1[as.vector(select),]`

Spatial* + Raster: `extract([raster obj], [SpatialPolygons object])`

Geometric Operations

Geometric Intersect: `gIntersect(SpatialObject1,SpatialObject2,byid=TRUE)`
Buffer: `gBuffer([SpatialPoints obj], width= [radius in projected units])`

Installation

Packages:

- `sp`: tools for vector spatial data
- `rgeos`: tools for distance operations (near, within, etc.)
- `rgdal`: tools for reading and writing files in different formats

Installation:
Update R to version > 3.1.
On Windows:

- `install.packages("sp"`
- `install.packages("rgdal")`
- `install.packages("rgeos")`

On OSX:

- `install.packages(c("sp","raster"))`
- Download and install GDAL Complete
- Download rgdal package.
- Open .dmg file and place `rgdal_0.9-1.tgz` on desktop.
- Run `install.packages("~/Desktop/rgdal_0.9-1.tgz",repos=NULL)`
- Run `install.packages("rgeos", type = "source", configure.args = "--with-geos-config=/Library/Frameworks/GEOS.framework/Versions/Current/unix/bin/geos-config")`

Open-Source GIS Software Acronyms

GIS tools in R are based on a set of tools developed by the open-source community and which underlie a great many GIS tools beside those available in R, including tools in Python and several stand-alone applications (likeQGIS). As a result, there are a number of acronyms youre likely to find if you start googling GIS tools heres a quick guide to them.

- **OSGeo**: Open-Source Geospatial Foundation; the group the manages the ecosystem of open-source GIS software.
- **GDAL**: Geospatial Data Abstraction Library. Once upon a time, OSGeo published two sets of tools OGRfor working with vector data, and GDAL for working with raster data. In recent years, however, these toolshave converged, so GDAL usually used to refer to the full library created by OSGeo. In R, however, the older meanings often still apply, which is why readGDAL() is for reading raster data and readOGR() is for reading vector data.
- **OGR**: OpenGIS Simple Features Reference (I think?). At one time OGR was the set of tools published byOSGeo for manipulating vector data. OGR is now officially a part of GDAL (which is why it comes in thergdal library).
- **proj4**: proj4 is standard format for describing projections.

- **GEOS**: Geometry Engine, Open Source. Set of tools for creating buffers, intersects, etc. the `rgeos` library is just a way of interfacing between R and this extremely fast and well-developed C++ library.
- **GRASS**: An OSGeo platform designed to unify the GDAL tools in a graphical user interface.¢ QGIS: An open-source program designed specifically to be an alternative to ArcGIS based on the GDALlibrary.
  Want to know more? Check out the OSGeo FAQs!