GIS analysis in R

# Introduction

It is now relatively easy to carry out GIS analyses in R, when before most people used plug-ins (some of which require a fee) and model-building in software such as ArcGIS.

With a combination of QGIS and R, the input and analysis of spatial data is completely free. Having the data extraction scripted in R also allows seamless integration with statistical analysis. In the past, this was possible in Python, but with R having the same capabilities (that is also getting increasingly user-friendly), one does not need to learn yet *another* programming language.

## Libraries

These are some of the libraries needed to load and analyse GIS layers in R.

library(rgdal) # Also loads the {sp} package

## Warning: package 'rgdal' was built under R version 3.4.4

## Loading required package: sp

## Warning: package 'sp' was built under R version 3.4.4

## rgdal: version: 1.2-18, (SVN revision 718)  
## Geospatial Data Abstraction Library extensions to R successfully loaded  
## Loaded GDAL runtime: GDAL 2.2.3, released 2017/11/20  
## Path to GDAL shared files: C:/R-3.4.3/library/rgdal/gdal  
## GDAL binary built with GEOS: TRUE   
## Loaded PROJ.4 runtime: Rel. 4.9.3, 15 August 2016, [PJ\_VERSION: 493]  
## Path to PROJ.4 shared files: C:/R-3.4.3/library/rgdal/proj  
## Linking to sp version: 1.2-7

library(rgeos)

## Warning: package 'rgeos' was built under R version 3.4.4

## rgeos version: 0.3-26, (SVN revision 560)  
## GEOS runtime version: 3.6.1-CAPI-1.10.1 r0   
## Linking to sp version: 1.2-7   
## Polygon checking: TRUE

library(raster)

## Warning: package 'raster' was built under R version 3.4.4

## Loading GIS layers

To read the GIS layer, you have to supply at least two arguments to the readOGR() function.

points.ly<-readOGR(".//GIS", "sampling points")

## OGR data source with driver: ESRI Shapefile   
## Source: "C:\Dropbox\Documents To Go\R sharing\GIS in R\GIS", layer: "sampling points"  
## with 7 features  
## It has 2 fields

trees.ly<-readOGR(".//GIS", "trees")

## OGR data source with driver: ESRI Shapefile   
## Source: "C:\Dropbox\Documents To Go\R sharing\GIS in R\GIS", layer: "trees"  
## with 2316 features  
## It has 1 fields

roads.ly<-readOGR(".//GIS", "roads")

## OGR data source with driver: ESRI Shapefile   
## Source: "C:\Dropbox\Documents To Go\R sharing\GIS in R\GIS", layer: "roads"  
## with 16 features  
## It has 1 fields  
## Integer64 fields read as strings: Lanes

shrubs.ly<-readOGR(".//GIS", "shrubs")

## OGR data source with driver: ESRI Shapefile   
## Source: "C:\Dropbox\Documents To Go\R sharing\GIS in R\GIS", layer: "shrubs"  
## with 1621 features  
## It has 1 fields

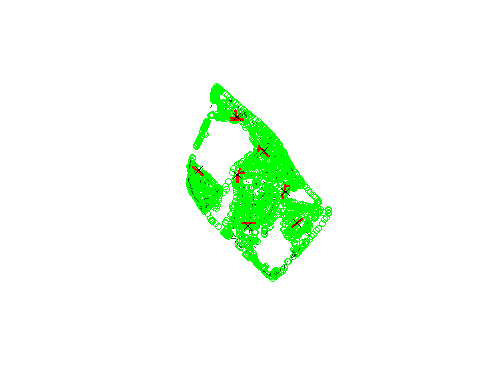
With ESRI .SHP files, the first argument is the directory where your files are contained, so you may have to edit this a bit depending on how you stored the files. The second argument would be the filename(s) for the set of shapefiles, without the extension(s), i.e., “.shp”.

Depending on the format of your GIS layer (i.e., GoogleEart .KML files instead of .SHP files, etc.), the two arguments vary a little in their required inputs.

## Visualization

Just as you would do a quick look at datasets you read into R, e.g., with the head() or str() functions, you should take quick look at the GIS layers you read–by plotting a quick map! This is just using the generic graphical function plot(); to plot a series of layers in the same plot, subsequent plot() calls need the add = TRUE argument.

plot(trees.ly, pch = 1, col = "green")  
plot(shrubs.ly, add = TRUE, col = "darkgreen", border = NA)  
plot(roads.ly, add = TRUE, col = "red", lwd=2)  
plot(points.ly, pch = 4, add = TRUE)



# Introduction to spatial objects

The layers are read and saved as spatial objects, i.e., a set of *geometric features* with a *projection system*. It may also have a data.frame of *attributes* for each feature.

## Projection system

The projection system provides a reference system so that we can know the position of any object relative to each other and also anything else on the surface of the Earth.

A good practice is to *always* check the projection system immediately after reading a GIS file.

proj4string(points.ly); proj4string(trees.ly); proj4string(shrubs.ly); proj4string(roads.ly)

## [1] "+proj=utm +zone=48 +datum=WGS84 +units=m +no\_defs +ellps=WGS84 +towgs84=0,0,0"

## [1] "+proj=utm +zone=48 +datum=WGS84 +units=m +no\_defs +ellps=WGS84 +towgs84=0,0,0"

## [1] "+proj=utm +zone=48 +datum=WGS84 +units=m +no\_defs +ellps=WGS84 +towgs84=0,0,0"

## [1] "+proj=utm +zone=48 +datum=WGS84 +units=m +no\_defs +ellps=WGS84 +towgs84=0,0,0"

They’re in the projection system we want, which is good. Otherwise, you would have had to choose one of the layers to be the projection system for all the other layers, and transform everything to it, i.e., the “on-the-fly projection” feature in QGIS.

proj\_utm<-proj4string(points.ly)  
trees.ly<-spTransform(trees.ly, CRS = CRS(proj\_utm))  
canopy.ly<-spTransform(shrubs.ly, CRS = CRS(proj\_utm))  
road.ly<-spTransform(roads.ly, CRS = CRS(proj\_utm))

The Universal Transverse Mercator (UTM) coordinate reference system is usually preferred, because the coordinates allow direct calculation of distances, area, etc., in metres. For Singapore, our zone/grid is 48N and WGS84.

## Geometries

There are three basic types of geometries: *points* (zero dimensions), *lines* (each segment one-dimensional, although a series of lines joined from end to end, i.e., a *polyline*, can be spread over two or more dimensions), and *polygons* (two-dimensional). It is possible to have three dimensions, i.e., a polyhedral, but we don’t have to worry about that now, especially when Singapore is relatively flat with little variation in altitude.

Therefore, there are also three types of spatial objects: SpatialPoints, SpatialLines, and SpatialPolygons. Spatial objects are essentially a collection of geometries with a coordinate reference system.

Just as with vectors, etc., you can check the class() and length() of a spatial object. The length is the number of features in the object.

class(points.ly); class(trees.ly); class(shrubs.ly); class(roads.ly)

## [1] "SpatialPointsDataFrame"  
## attr(,"package")  
## [1] "sp"

## [1] "SpatialPointsDataFrame"  
## attr(,"package")  
## [1] "sp"

## [1] "SpatialPolygonsDataFrame"  
## attr(,"package")  
## [1] "sp"

## [1] "SpatialLinesDataFrame"  
## attr(,"package")  
## [1] "sp"

length(points.ly); length(trees.ly); length(shrubs.ly); length(roads.ly)

## [1] 7

## [1] 2316

## [1] 1621

## [1] 16

# Attributes

Each feature can then have attributes. In ArcGIS and QGIS, there is the attribute table. Likewise in R, and can be accessed with the @data slot.

head(points.ly@data)

## Site Point\_ID  
## 1 Queenstown 5  
## 2 Queenstown 4  
## 3 Queenstown 3  
## 4 Queenstown 6  
## 5 Queenstown 7  
## 6 Queenstown 2

head(trees.ly@data)

## Species  
## 1 Xanthostemon chrysanthus  
## 2 Xanthostemon chrysanthus  
## 3 Peltophorum pterocarpum  
## 4 Peltophorum pterocarpum  
## 5 Peltophorum pterocarpum  
## 6 Peltophorum pterocarpum

head(shrubs.ly@data)

## Species  
## 0 Bougainvillea cultivar  
## 1 Polyscias fruticosa  
## 2 Hymenocallis speciosa  
## 3 Hymenocallis speciosa  
## 4 Polyscias fruticosa  
## 5 Ixora cultivar

head(roads.ly@data)

## Lanes  
## 0 2  
## 1 1  
## 2 1  
## 3 2  
## 4 1  
## 5 1

The no. of lanes is a factor for some reason, so we want to convert it to a number to use in calculations later. Note to BIRT team: is it also just *one side* of the road? If so, I’m multiplying by two here.

roads.ly$Lanes<-as.numeric(as.character(roads.ly$Lanes))\*2

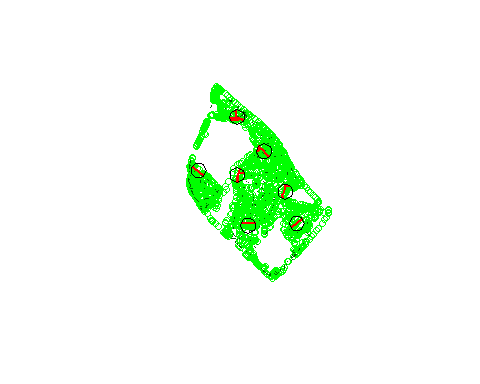
# Basic GIS functions

In this data set, there are seven sampling points.

## Buffers

Let’s say we want to extract landscape data from a circular area, i.e., a *buffer*, within a radius of 50 m of each point.

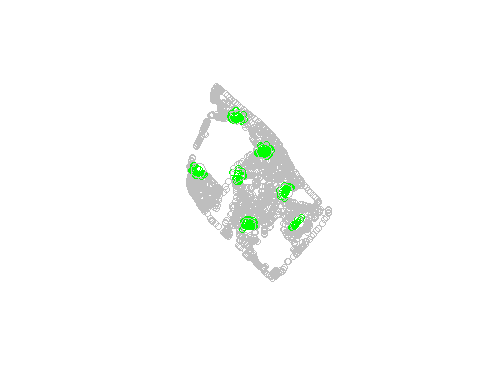
points\_buffer50<-gBuffer(points.ly, byid = TRUE, width = 50)  
  
plot(trees.ly, pch = 1, col = "green")  
plot(shrubs.ly, add = TRUE, col = "darkgreen", border = NA)  
plot(roads.ly, add = TRUE, col = "red", lwd=2)  
plot(points\_buffer50, add = TRUE)



As you can see, we will have to find what is within the 50-m radius buffer (which, remember, is a polygon) before extracting the environmental variables. Let’s try this with trees (which are points).

The function intersect() from the {raster} package finds the intersection between the two geometries provided, and combines the attributes from both spatial objects for each feature.

trees\_buffer50<-intersect(trees.ly, points\_buffer50)  
  
plot(trees.ly, col = "gray", pch=1)  
plot(trees\_buffer50, col="green", pch = 1, add = TRUE)  
plot(points\_buffer50, border = "gray", add = TRUE)



head(trees\_buffer50)

## Species Site Point\_ID  
## 1 Xanthostemon chrysanthus Queenstown 3  
## 16 Xanthostemon chrysanthus Queenstown 3  
## 17 Xanthostemon chrysanthus Queenstown 3  
## 18 Xanthostemon chrysanthus Queenstown 3  
## 19 Xanthostemon chrysanthus Queenstown 3  
## 20 Xanthostemon chrysanthus Queenstown 3

Say we want the number of trees of each tree species within each 50-m buffer.

table(trees\_buffer50$Point\_ID)

##   
## 1 2 3 4 5 6 7   
## 29 75 27 20 25 17 28

We can also cross-tabulate the Point\_ID with the tree Species.

trees\_buffer50.com<-xtabs(~Point\_ID+Species, data=trees\_buffer50@data)  
trees\_buffer50.com[,1:6]

## Species  
## Point\_ID Acacia auriculiformis Acacia mangium Adenanthera pavonina  
## 1 0 0 2  
## 2 0 0 0  
## 3 0 0 0  
## 4 0 0 0  
## 5 0 0 0  
## 6 0 0 0  
## 7 0 0 0  
## Species  
## Point\_ID Albizia saman Aleurites moluccana Alstonia scholaris  
## 1 0 0 7  
## 2 0 0 5  
## 3 3 0 0  
## 4 0 0 0  
## 5 0 0 0  
## 6 0 0 4  
## 7 0 0 0

This is like a community matrix; so we can extract, e.g., the number of tree species, or diversity index of trees, using functions from the library {vegan}.

library(vegan)

## Loading required package: permute

## Loading required package: lattice

## This is vegan 2.4-5

specnumber(trees\_buffer50.com)

## 1 2 3 4 5 6 7   
## 12 10 6 7 4 5 6

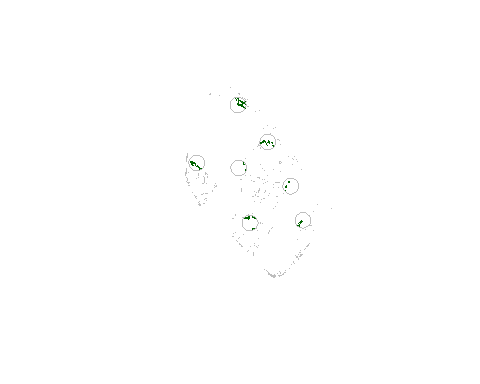
diversity(trees\_buffer50.com, index = "shannon")

## 1 2 3 4 5 6 7   
## 2.1922617 1.6753442 1.5874771 1.6424615 0.8605493 1.5643701 1.5691529

These can all be used as variables in statistical analysis!

Let’s now try doing this with shrubs, i.e., polygons over polygons.

shrubs\_buffer50<-intersect(shrubs.ly, points\_buffer50)  
  
plot(shrubs.ly, col="gray", border=NA)  
plot(shrubs\_buffer50, col="darkgreen", border="darkgreen", add = TRUE)  
plot(points\_buffer50, border = "gray", add = TRUE)



I want the shrub polygons to be *clipped* so that those areas outside the buffer are excluded.

If the information we want about the shrubs are related to the area, we need to calculate the area of each shrub polygon.

shrubs\_buffer50$Area<-gArea(shrubs\_buffer50, byid = TRUE)  
  
head(shrubs\_buffer50@data)

## Species Site Point\_ID Area  
## 1 Polyscias fruticosa Queenstown 2 5.077178  
## 2 Ixora cultivar Queenstown 2 3.434199  
## 3 Pandanus pygmaeus Queenstown 2 4.991320  
## 4 Ixora cultivar Queenstown 2 19.611219  
## 5 Ixora cultivar Queenstown 2 24.623360  
## 6 Wrightia religiosa Queenstown 2 4.852919

You can now get the total area of shrubs within the buffer.

with(shrubs\_buffer50@data, tapply(Area, Point\_ID, sum))

## 1 2 3 4 5 6 7   
## 352.54239 150.94736 23.56300 27.76144 140.97538 43.47500 205.72323

And also cross-tabulate shrub Species by Point\_ID to calculate diversity, etc.

shrubs\_buffer50.com<-xtabs(Area~Point\_ID+Species, data=shrubs\_buffer50)  
  
diversity(shrubs\_buffer50.com, index = "shannon")

## 1 2 3 4 5 6 7   
## 1.8792370 1.4200191 0.9733143 2.1182169 1.7200863 1.2321293 1.4067496

Finally, with lines. Say we want to calculate an index of road lane density *sensu* Chong et al. (2014; *Biological Conservation*),

We need to

1. Clip the roads to the buffer, if it has not been done yet;
2. Calculate the length of each road segment with the gLength() function;
3. Calculate where is the number of road lanes in each road segment;
4. Divide by the area of the buffer, .

roads\_buffer50<-intersect(roads.ly, points\_buffer50)  
  
roads\_buffer50@data$length<-gLength(roads\_buffer50, byid=TRUE)  
  
lengthxlanes<-with(roads\_buffer50@data, Lanes \* length)  
lengthxlanes.sum<-tapply(lengthxlanes, roads\_buffer50@data$Point\_ID, sum)  
  
lengthxlanes.sum/(pi\*50^2)

## 1 2 3 4 5 6   
## 0.08074669 0.04427733 0.06612658 0.02522000 0.04855520 0.07168019   
## 7   
## 0.02502442

The units is m/m2.