Applied Geo-Scripting: Lesson 1

Jan Verbesselt, Sytze de Bruyn, Loïc Dutrieux, Ben De Vries, Aldo Bergsma

October 21, 2013

Geo-scripting learning objectives

- Handle spatial data using a scripting language
- Read, write, and visualize spatial data (vector/raster) using a script
- Know how to find help (on spatial data handling functions)
- Solve scripting problems (debug, reproducible example, writing functions)
- Find libraries which offer spatial data handling functions
- Learn to include functions from a library in your script
- Apply learned concepts in a case study: learning how to address a spatial/ecological/applied case (e.g. detect forest changes, ocean floor depth analysis, bear movement, etc.) with a raster and vector dataset.

Today's topics

- Intro to basic concepts of applied scripting for spatial data
- Why geo-scripting?
- Course planning and practical issues
- Getting up to speed with R and loading 'rasta' package

RASTA: Reproducible and Applied Spatial and Temporal Analaysis
http://rasta.r-forge.r-project.org

Why geo-scripting?

- Reproducible: avoid clicking and you keep track of what you have done
- Efficient: you can write a script to do something for you e.g. multiple times e.g. automatically downloading data
- Build your own tools and functions (e.g. raster filters, MODIS download tool, BFAST package)
- Enable collaboration: sharing scripts, functions, and packages
- Good for finding errors i.e. debugging e.g. this course is fully writting with scripting languages (i.e. R and Latex).

What is a scripting language?

- A scripting language or script language is a programming language that supports the writing of scripts, programs written for a special runtime environment that can interpret and automate the execution of tasks which could alternatively be executed one-by-one by a human operator
- Different from compiled languages like C/C++/Fortran.
- A scripting language is the glue, between different commands, functions, and objectives without the need to compile it for each OS/CPU Architecture

Different scripting languages for geo-scripting

The main scripting languages for GIS and Remote sensing currently are:

- R
- Python
  - stand-alone (ArcPy and PyQGIS)
  - integrated within ArcGIS, QGIS
- GRASS (grass function are included in QGIS)
- Javascript for geoweb scripting

Python versus R

- Python is a general purpose programming language with a clear syntax
- R is particularly strong in statistical computing and graphics
- Installing libraries in Python is sometimes challenging
- Syntactic differences can be confusing (especially when combining these)
- There are many R and Python packages for spatial analyses and for dealing with spatial data
- Scripts in both languages can be combined:
  - call R from Python using **RPy, RPy2**
  - call Python from R `http://rpython.r-forge.r-project.org/`
- Many programs have support for:
  - Python (Blender, Sketchup, QGIS, MySQL, PostGIS)
  - R (GRASS, QGIS, MySQL, PostGIS)

Course set-up

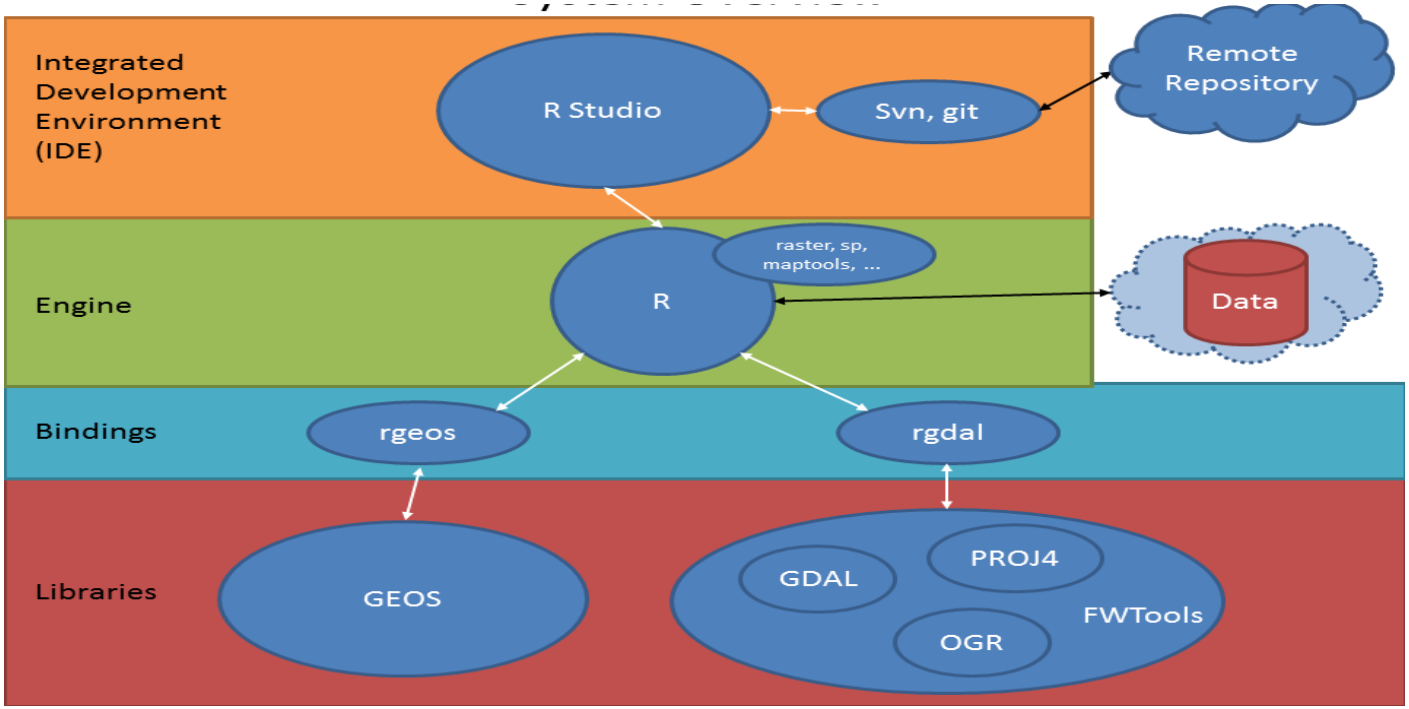

Figure 1 : Course set-up

- SVN (SubVersioN): Version control system for scrits and docs
- R libraries: rgeos, rgdal
- GDAL: Geospatial Data Abstraction Library http://www.gdal.org/
- GEOS: Geometry Engine, Open Source
  http://trac.osgeo.org/geos/

Get Your R On

Getting started with Rstudio This preliminary section will cover some basic details about R. For this course we will use Rstudio as an IDE to write and run scripts. Open Rstudio! Now type the following script in the R console:

```
>rm(list=ls())  # Clear the workspace!
>ls() ## no objects left in the workspace
```

```
character(0)
```

A good way to start most R scripts

```
>a <- 1
>a
```

```
[1] 1
```

The first line you passed to the console created a new object named *a* in memory. The symbol '<-' is somewhat equivalent to an equal sign. In the second line you printed *a* to the console by simply typing it's name.

What is the class of this object?

Get Your R On

```
>class(a)
```

```
[1] "numeric"
```

You now have requested the **class** attribute of *a* and the console has returned the attribute: **numeric**. R possesses a simple mechanism to support an object-oriented style of programming. All objects (*a* in this case) have a class attribute assigned to them. **R** is quite forgiving and will assign a class to an object even if you haven't specified one (as you didn't in this case). Classes are a very important feature of the **R** environment. Any function or method that is applied to an object takes into account its class and uses this information to determine the correct course of action.

Custom Functions

It is hard to unleash the full potential of R without writing your own functions. Luckily it's very easy to do. Here are some trivial examples:

```
>add <- function(x){
+ #put the function arguments in () and the evaluation in {}
+   x + 1
+ }
>add(4)
```

```
[1] 5
```

```
>#Set the default values for your function--
>add <- function(x = 5){
+   x + 1
+ }
>add()  #automatically evaluates x = 5
```

```
[1] 6
```

```
>add(6) #but you can still change the defaults
```

```
[1] 7
```

That's about all there is too it. The function will generally return the result of the last line that was evaluated. However you can also use return() to

Now, let's declares a new object, a new function, **newfunc** (this is just a name and if you like you can give this function another name). Appearing in the first set of brackets is an argument list that specifies (in this case) two names. The value of the function appears within the second set of brackets where the process applied to the named objects from the argument list is defined.

```
>newfunc <- function(x, y) {
+   2*x + y
+ }
>a2b <- newfunc(2, 4)
>a2b
```

```
[1] 8
```

```
>rm(a, newfunc, a2b)
```

Help?!

R is supported by a very comprehensive help system. Help on any function can be accessed by entering the name of the function into the console preceded with a ?. The easiest way to access the system is to open a web-browser. This help system can be started by entering **help.start()** in the R console. Try it and see what happens.

Data Structures

There are several ways that data are stored in R. Here are the main ones:

- **Vectors** The most generic data structure. In R, any variable of an atomic data type (numeric, integer, logical, character) is a vector. This will be examplified below.
- **Data Frames** The most common format. Similar to a spread sheet. A data.frame() is indexed by rows and columns and store numeric and character data. The data.frame is typically what we use when we read in csv files, do regressions, et cetera.
- **Matrices and Arrays** Similar to data.frames but slightly faster computation wise while sacrificing some of the flexibility in terms of what information can be stored. In R a matrix object is a special case of an array that only has 2 dimensions. i.e., an array is n-dimensional matrix while a matrix only has rows and columns (2 dimensions)
- **Lists** The most common and flexible type of R object. A list is simply a collection of other objects. For example a regression object is a list of: 1)Coefficient estimates 2) Standard Errors 3) The Variance/Covariance matrix ...

We will look at examples of these objects in the next sectionl

R packages and the rasta package

R 'packages' are user contributed functions. There are about 5000 or so (with a constantly expanding list). If a package is already installed you load the package with the `library()` command. If you want to install a package you can use the `install.packages()` command (you have to provide the url of the CRAN mirror to download the package. If you are using R Studio you can also just click on **Tools>Install Packages**, and type in the name(s) of the package you want to install.
Now install and load the rasta package:

```
>install.packages("rasta", repos="http://R-Forge.R-project.org")
```

```
>library(rasta) ## load the rasta library
```

```
># ?mysummary
>mysummary
```

What does the function do?

Reading Data in and Out

The most common way to read in data is with the `read.csv()` command. Type ?read.table in your console for some other examples.

```
>f <- system.file("extdata/kenpop89to99.csv", package="rasta")
>mydat<-read.csv(f)
```

We can explore the data using the names(), summary(), head(), and tail() commands (we will use these frequently through out the exercise)

```
>names(mydat)[1:3] #colunm names
```

```
[1] "ip89DId"   "ip89DName" "ADMIN3"
```

```
>summary(mydat$Y89Pop)[1:3]
```

```
   Min. 1st Qu.  Median
  57960  222900  451500
```

```
>head(mydat$Y89Births)[1:2]
```

```
[1] 42560 27720
```

What is the class of the *mydat*? We will go over ways to index and subscript data.frames later. Lets do a basic regression so you can see an example of a list

Basic regression and example of a list

We use the `lm()` command to do a basic linear regression. The ~ symbol separates the left and right hand sides of the equation and we use '+' to separate terms and '*' to specify interactions. *Regress the Population in 1999 on the population and birthrate in 1989*

```
>myreg<-lm(Y99Pop ~ Y89Births + Y89Brate, data = mydat)
>myreg[c(1,8)]
```

```
$coefficients
(Intercept)   Y89Births    Y89Brate
502592.5928     38.0455 -14369.0931

$df.residual
[1] 45
```

Basic regression and example of a list

A regression object is an example of a list. We can use the `names()` command to see what the list contains. We can use the summary() command to get a standard regression output (coefficients, standard errors, et cetera) and we can also create a new object that contains all the elements of a regression summary.

```
>names(myreg)[1:3]
```

```
[1] "coefficients" "residuals"    "effects"
```

```
>myregsum <- summary(myreg)
>names(myregsum)[1:2]
```

```
[1] "call"  "terms"
```

```
>myregsum[['adj.r.squared']] #extract the adjusted r squared
```

```
[1] 0.8937546
```

```
>myregsum$adj.r.squared # does the same thing
```

```
[1] 0.8937546
```

why is *myregsum* a list object? What is the advantage of a list? That

Set Your Working Directory

Let's do some basic set up first.

- Create a folder which will be your working directory e.g. *Lesson1*
- Create an R script within that folder
- Set your working directory to the *Lesson2* folder
- Create a *data* folder within your working directory

In the code block below type in the file path to where your data is being held and then (if you want) use the setwd() (set working directory) command to give R a default location to look for data files.

```
>setwd("yourworkingdirectory")
># This sets the working directory (where R looks for files)
>getwd() # Double check your working directory
```

```
>datdir <- file.path("data") ## path
```

Ex.Lesson 1: Write you own function to create a spatial map a country

Write a function to visualise data and plots different variables

- Submit a clear, reproducible, and documented script to create a spatial map
- The script needs to contain your name
- Filename: *lastnamefirstname.R*

This will be a test to see if your R scripting levels are ok to continue the course in the coming months.

```
>## Load required packages
>library(raster)
```

```
>## Download data from gadm.org
>adm <- getData('GADM', country='PHL', level=2, path = datdir)
```

Ex.Lesson 1: Write you own function to create a spatial map a country

```
>mar <- adm[adm$NAME_1=="Marinduque",]
>plot(mar, bg="dodgerblue", axes=T)
>plot(mar, lwd=10, border="skyblue", add=T)
>plot(mar, col="green4", add=T)
>grid()
>box()
>invisible(text(getSpPPolygonsLabptSlots(mar),
+ labels=as.character(mar$NAME_2), cex=1.1, col="white", font=2))
>mtext(side=3, line=1, "Provincial Map of Marinduque", cex=2)
>mtext(side=1, "Longitude", line=2.5, cex=1.1)
>mtext(side=2, "Latitude", line=2.5, cex=1.1)
>text(122.08,13.22, "Projection: Geographic\n
+ Coordinate System: WGS 1984\n
+ Data Source: GADM.org\n
+ Created by: ARSsalvacion", adj=c(0,0), cex=0.7, col="grey20")
```

Ex.Lesson 1: Write you own function to create a spatial map a country

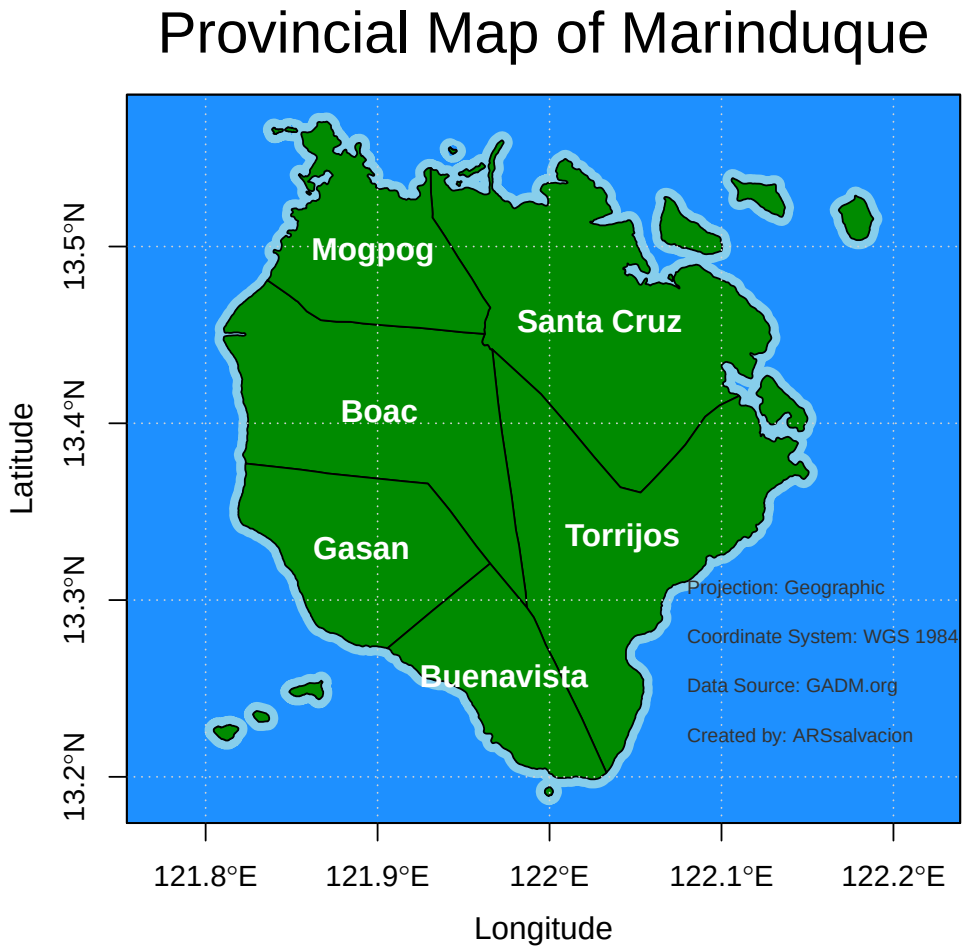

Figure 2 : Adminstrative boundaries of Marinduque

More information

For more information about R please refer to the following links:

- http://www.statmethods.net/index.html This is a great website for learning R function, graphs, and stats.
- the book on Applied spatial Data analysis with R http://www.asdar-book.org/ (Bivand et al., 2013).
- Visit http://www.r-project.org/ and check out the Manuals i.e an introductions to R
- Overview of R functionality for spatial data analysis: http://cran.r-project.org/web/views/Spatial.html
- http://gis.stackexchange.com/questions/45327/tutorials-to-handle-spatial-data-in-r

Extra challenge - for ggplot lovers

Optional challenge

```
>require(ggmap)
>shp.spdf <- adm[adm$NAME_1=="Marinduque",]
>shp.df <- fortify(shp.spdf)
>shp.centroids.df <- data.frame(long = coordinates(shp.spdf)[,1],
+                                lat = coordinates(shp.spdf)[,2])
># Get names and id numbers corresponding to administrative areas
>shp.centroids.df[, 'ID_1'] <- shp.spdf@data[,'ID_1']
>shp.centroids.df[, 'NAME_2'] <- shp.spdf@data[,'NAME_2']
>q <- qmap(location = "Marinduque", zoom = 10, maptype = "satellite")
>q = q +  geom_polygon(aes(x = long, y = lat, group = group),
+     data = shp.df,
+     colour = 'white', fill = 'black', alpha = .4, size = .3)
>q = q + geom_text(data = shp.centroids.df,
+     aes(label = NAME_2, x = long, y = lat, group = NAME_2),
+                  size = 3, colour= "white")
>print(q)
```

Extra challenge - for ggplot lovers

Figure 3 : GoogleMap Satellite view of Marinduque

Bivand, R. S., Pebesma, E. J., & Rubio, V. G. (2013). Applied spatial data analysis with R, .