Lesson 5: Introduction to raster based analysis in R

Loïc Dutrieux

October 15, 2013

1 Today's learning objectives

At the end of the lecture, you should be able to:

- Read/write raster data
- Perform basic raster file operations/conversions
- Perform simple raster calculations

2 Assumed knowledge from previous lectures

- Understand system architecture (R, R packages, libraries, drivers, bindings)
- Read/write vector data
- Good scripting habits

3 Reminder on overall system architecture

In a previous lecture you saw the overall system architecture (Figure 1) and you saw how to read/write vector data from/to file into you R environment. These vector read/write operations were made possible thanks to the OGR library. The OGR library is interfaced with R thanks to the rgdal package/binding. By analogy, raster data can be read/written thanks to the GDAL library. Figure 1 provides an overview of the connections between these elements. GDAL stands for Geospatial Data Abstraction Library. You can check the project home page at `http://www.gdal.org/` and you will be surprised to see that a lot of the software you have used in the past to read gridded geospatial data use GDAL. (i.e.: ArcGIS, QGIS, GRASS, etc). In this lesson, we will use GDAL indirectly via the raster package, which uses rgdal extensively. However, it is possible to call GDAL functionalities directly through the command line (a shell is usually provided with the GDAL binary distribution you installed on your system), which is equivalent to calling a system() command directly from within R. In addition, if you are familiar with R and its string handling utilities, it should facilitate ...

[1] Note that FWTools is one example of binary distribution for windows, you can also install gdal/ogr/proj.4 from OSGeo4W, (linux: by compiling it yourself from source or from a package archive)

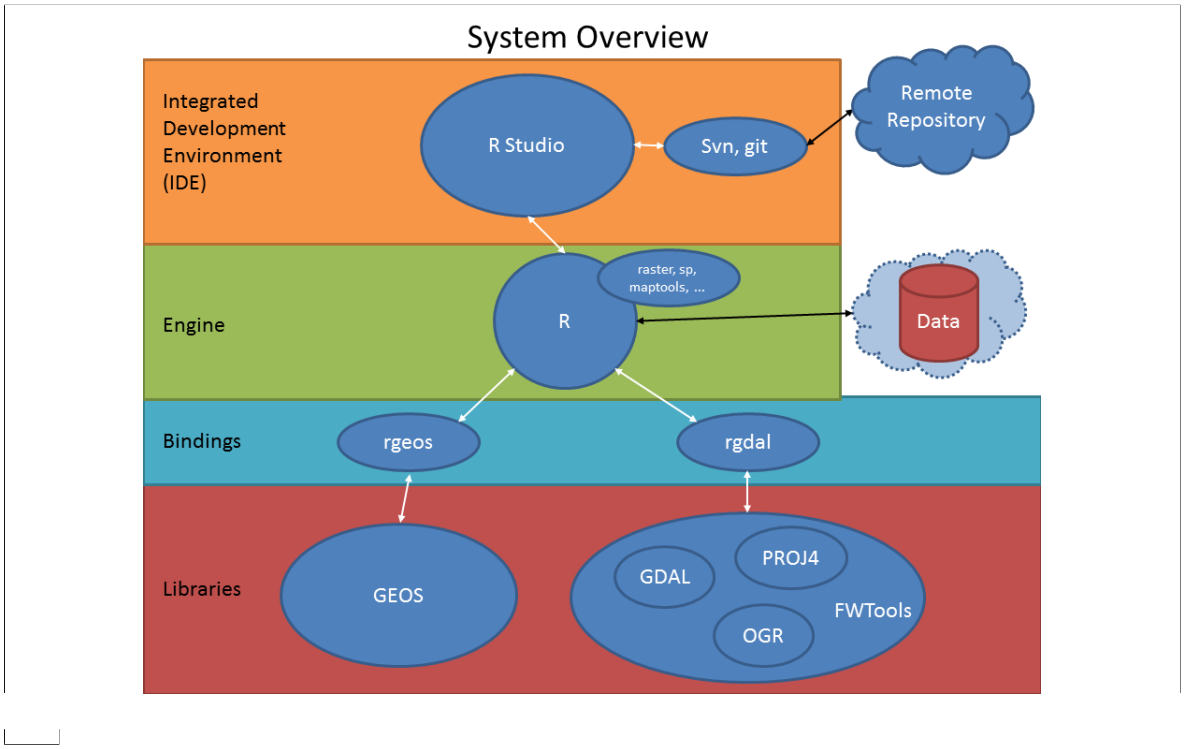

Figure 1: System architecture overview[1]

4 overview of the raster package

The raster package is the reference R package for raster processing, Robert J. Hijmans is the original developer of the package. The introduction of the raster package to R has been a revolution for geo processing and analysis. Among other things the raster package allows to:

- Read and write raster data of most commonly used format (thanks to extensive use of rgdal)
- Perform most raster operations (creation of raster objects, performing spatial/geometric operations (re-projections, resampling, etc), filtering and raster calculations)
- Work on large raster datasets thanks to its built-in block processing functionalities
- Perform fast operations thanks to optimized back-end C code
- Visualize and interact with the data
- etc...

Check the home page of the raster package `http://cran.r-project.org/web/packages/raster/`, the package is extremely well documented, including vignettes and demos.

Read/write raster data into R using the raster package

Explore the raster objects

The raster package produces and uses R objects of three different classes. The **RasterLayer**, the **RasterStack** and the **RasterBrick**. A RasterLayer is the equivalent of a single layer raster, as an R environment variable. The data themselves, depending on the size of the grid can be loaded in memory or on disk. The same stands for RasterBrick and RasterStack objects, which are the equivalent of multi-layer RasterLayer objects. RasterStack and RasterBrick are very similar, the difference being in the virtual characteristic of the RasterStack. While a RasterBrick has to refer to one multi-layer file or is in itself a multi-layer object with data loaded in memory, a RasterStack may "virtually" connect several raster objects written to different files or in memory. Processing will be more efficient for a RasterBrick than for a RasterStack, but RasterStack has the advantage of facilitating pixel based calculations on separate raster layers.

Let's take a look into the structure of these objects.

```
> library(raster)
> r <- raster(ncol=20, nrow=20)
> class(r)

[1] "RasterLayer"
attr(,"package")
[1] "raster"

> # Simply typing the object name displays its general properties / metadata
> r

class       : RasterLayer
dimensions  : 20, 20, 400  (nrow, ncol, ncell)
resolution  : 18, 9  (x, y)
extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84
```

5 Raster objects manipulations

Re-projections

Creating layer stacks

6 Simple raster arithmetic

Vectorized expressions
Calc and overlay
Examples:
Perform cloud replacement

1. vectorized

2. with calc();

Calculate NDVI Zonal statistics (using vector layer as input)? Ben?

7 Hdf files (Not part of the course)

A file format that is getting increasingly common with geo-spatial gridded data is the hdf format. Hdf stands for hierarchical data format. For instance MODIS data have been delivered in hdf format since its beginning, Landsat has adopted the hdf format for delivering its Level 2 surface reflectance data recently. This data format has an architecture that makes it very convenient to store data of different kind in one file, but requires slightly more effort from the researcher to work with conveniently.

Windows

The rgdal package does not include hdf drivers in its pre-built binary. Data can therefore not be read directly from hdf into R (as object of class rasterLayer). A workaround is to convert the files externally to a different data format. Since you probably have gdal installed on your system (either via FWTools or OSGeo4W), you can use the command line utility gdal_translate to perform this operation. One easy way to do that is by calling it directly from R, via the command line utility.

TODO(dutri001) add a not run example

Linux

8 Further reading

9 Packages you should know about