A collection of tips for R in Finance

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. R is a great statistical software by any measure. Its popularity has grown dramatically over the past few years going from being used almost exclusively in academia to a huge community of users across all fields. Today it's probably the most widely used statistical software and it offers the richest list of statistical routines developed by an ever growing community of R enthusiasts. Interested readers can have a look at a recent study comparing the popularity of data analysis software. This is compelling to say the least.

I often red and heard that R learning curve is steep so I decided to write this short article which will help a few people hopefully. The goal is to put together some simple tools, methods and tips to create an efficient development environment.

First things first: you need a Code Editor

After a very short while with R you will probably feel that using the R Console only is very tedious and you will want a proper code editor. There is an excellent list of IDE/Text Editors on R website. I tested pretty much all of those available on Windows and after a few years of using XEmacs (which is excellent by the way), I currently use Notepad++. The main advantage over XEmacs is its lightness and the ease of installation for Windows users. Andrew Redd has written an excellent plugin that allow submitting R code from Notepad++. You can download the plugin here

Notepad++ does not support auto-completion for the R language straight out of the box. Fortunately Yihui Xi has written a small piece of code that allows auto-completion. The following is based on what can be found on his blog. If you want to use the full power of this tool, I strongly advise to load first all the R packages you need and then run the below piece of code:

```
source('http://yihui.name/en/wp-
content/uploads/2010/08/Npp_R_Auto_Completion.r')

# R.xml will be generated under your current work directory: getwd()
```

A file called R.xml will be created on your working directory (type getwd() at the R prompt if you're not sure what is your working directory) . You just have to put that file into Notepad++ install directory. In addition to the base packages this will make available auto completion for all the packages you loaded.

Another interesting feature of Notepad++ is the ability to create macros. (`Macro -> Start Recording`). Two that I found particularly useful are the R assignment operator (<-) and one that displays the following code:

```
plot(data,type="l")
```

When working with time series it's always very useful to plot the data. The above code is displayed by hitting a single key (assuming you have a one key shortcut) each time you need to plot the data.

The only feature that I'm missing in Notepad++ (compared to XEmacs) is the ability to list all the functions available in a file. This is probably not a big job but I didn't have the time to look at it so far. Anyone wanting to contribute is very welcome!

Organizing R code

After a while you will start writing your own functions. Some are highly specialized but you might want to use other on a regular basis. What I do (and most long term R users probably do the same) is saving those functions on a separate file. The file can then be loaded each time a new project is started.

As an example of general purpose functions, Patrick Burns has written two short functions called *head* and *tail* that are extremely useful. Basically *head* just displays the first n lines of a dataframe or a matrix. *Tail* does the same for the last n lines. This is particularly useful when you want to inspect visually a dataframe without having to display it entirely (which happens to me all the time). Those functions and much more are available on Burns Statistics.

Update packages

There are several releases of R per year. Once you started to use it for a while you will probably get tired of re loading R packages each time you upgrade to a new version. This issue has been discussed a few times in the R mailing list and there are a few ways to handle it: here is mine. I simply save an R script that installs automatically the list of packages I need. Once I installed the new R version, I run the below script.

```
install.packages(c("chron","fBasics",…))
```

There are many options available with this command including the choice of repository, the dependency options, etc….

Getting external data into R

From my personal experience there is no better way than .csv files to load data into R. The format is dead simple, flexible and you can play around with fairly large datasets without (too much) trouble. R provides a simple function to load csv files.

```
data <- read.csv(…)
```

However you might have to interface with relational databases. In that case R provides a wide range of possibilities. RODBC is a package that connects R to SQL Server, Access or Excel. Amongst other features it allows to pass SQL code within R and get the result straight back onto an R object. By the way, I'm not a Windows fan just a pragmatic user….

There are equivalent packages providing interfaces to MySQL and PostgreSQL: RMySQL and RPostgreSQL. I haven't tested them but I never heard of bad comments from users.

Communicating with external applications

I red somewhere that R shouldn't be seen as an island but rather a specific tool to do specific tasks. I can't think of any better definition of my day to day job. R is unmatched when it comes to statistical analysis but you might sometimes need to call it from external applications or you might need to get data from external sources straight into R…. Whatever the task at hand there is a good chance that you will find just the tool you need. I list below some of the most useful tools I came across:

R-(D)COM is a programming interface to COM and DCOM (ex ActiveX; Microsoft distributed object interface) to access the R calculation engine. As such, it runs only under the Windows environment. Thomas Baier and Erich Neuwirth have created a website dedicated to this technology. You will find there detailed instructions about installation and a video with examples. Using the same technology Vincent Vella's blog explains how to make R works within C#.

The RBloomberg package developed by Robert Sams is another interesting tool. It allows essentially to retrieve data from Bloomberg and put them straight into R. More details are available on the CRAN and on R-Finance mailing list.

Obviously the list doesn't stop here but it's beyond the scope of this article to list all of them.

Optimizing R code

In typical financial applications R code doesn't really have to be optimized when you deal with daily or lower frequency data. However when dealing with intraday or (even worst) tick data writing proper R

code makes a huge difference. There are several R tools available out there to make R code better (i.e run faster). The code bellow tells how long it takes to run a given set of R instructions

```
System.time([your R code])
```

We can go further and identify which part of your code takes longer.

```
Rprof("d1.prof")

[your R code]

Rprof(NULL)

d1prof = summaryRprof("d1.prof")

head(d1prof$by.self)
```

The output of the above code will display the time taken by each command in absolute and relative terms.

```
               self.time self.pct total.time total.pct
[.data.frame        0.54     26.0       0.88      42.3
==                  0.28     13.5       0.28      13.5
&                   0.16      7.7       0.16       7.7
[[.data.frame       0.08      3.8       0.22      10.6
[                   0.06      2.9       0.94      45.2
which               0.06      2.9       0.74      35.6
match               0.06      2.9       0.24      11.5
inherits            0.06      2.9       0.14       6.7
!=                  0.06      2.9       0.06       2.9
.External           0.06      2.9       0.06       2.9
```

Once the slow parts identified the next step is to go back to the “drawing board” and start thinking about better ways of doing what you’re doing. From my personal experience here is a list of things to avoid when writing R code

- ***Loops***: they are a real “plague” in R. Use lapply, sapply, tapply … functions instead whenever possible.

- ***Which*** statement: very often a good deal of time is spent on this statement when I can't avoid using it
- ***datatype conversion***: is very time consuming. Avoiding converting from one type to another and using only numeric make usually things running faster.

Finally Matthew Dowle has written an excellent package called data.table that allows "fast subset, fast grouping and fast merge in a short and flexible syntax". The package provides a significant improvement in performance when working with large datasets.

Conclusion

The tools presented in this article are only a small sample of what is available in the R world and represent what I found to be the most useful. Obviously it depends largely on personal taste and the kind of task at hand but from my experience it is a rather efficient set up.

I didn't touch upon parallel computing which is high up on the list of priorities when dealing with large datasets. The keen reader can have a look at the large literature and the various packages on the topic (here)

Arnaud Amsellem

arnaud.amsellem@fisycscapital.com