DSC 2001 Proceedings of the 2nd International Workshop on Distributed Statistical Computing
March 15-17, Vienna, Austria
http://www.ci.tuwien.ac.at/Conferences/DSC-2001
K. Hornik & F. Leisch (eds.) *ISSN 1609-395X*

Embedding R in Standard Software, and the other way round

Erich Neuwirth[*] **Thomas Baier** [†]

Abstract

We will discuss embedding R in standard software like spreadsheets to bring powerful statistical methods closer to the end user. This embedding has to consider design issues, technical issues, and user interface issues which will be addressed in our paper.

1 Introduction

A close second to word processing software, spreadsheets are among the most widely available and used programs categories. Additionally, more and more statistical data sets "come to live" as spreadsheets, in most cases as Excel spreadsheet. Therefore, connecting spreadsheets and R can be very helpful for performing statistical analyses efficiently.

Furthermore, spreadsheet programs offer an interaction model radically different from an "enriched" programming language like R. Spreadsheet programs implement a direct manipulation interaction model, the user creates models and formulas by clicking and pointing. Additionally, spreadsheets offer facilities for creating user controllable animated graphics, which is (not yet) built into R itself. So, by connecting spreadsheets and R, we not only make spreadsheets more powerful, we also offer additional presentation opportunities for R.

The software connection we are presenting is made possible through the implementation of R as a DCOM-server. DCOM is the system wide object model mechanism for the Microsoft Windows family of operating systems. Therefore, this

[*]Department of Statistics and Decision Support Systems, University of Vienna
[†]Department of Statistics, Vienna University of Technology

mechanism can be used by by any DCOM-enabled program. This includes all of the Microsoft office programs, and, e.g., any other program that uses VBA (Visual Basic for Applications) as the embedded macro language, and any Microsoft development system (Visual Basic, C, C++ ...).

On the other hand, DCOM is easily available *only* on Microsoft Windows, so our toolkit is platform specific.

2 Some examples

2.1 Basic interface

In the most basic interface, the user has a toolbar in Excel, and the toolbar looks like this:

Figure 1: User interface toolbar at startup

When the button `Start R` is clicked, it changes to

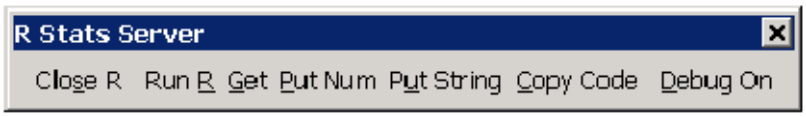

Figure 2: Activated user interface toolbar

The functions of this interface are rather simple. `Start R` starts the R demon in the background.

`Start R`	starts the R demon in the background
`Stop R`	stops the R demon in the background
`Run R`	runs R code from a spreadsheet range
`Get`	transfers a variable's value from R to a spreadsheet range
`Put Num`	transfers numeric values from a spreadsheet range to an R variable
`Put String`	transfers string values from a spreadsheet range to an R variable
`Copy code`	helps transforming code segments into encapsulated VBA modules
`Debug On`	shows code sent from Excel to R while being executed
`Debug Off`	switches debugging mode off

This mode gives the "raw interface" which might also be called *developer mode*. In this mode, the user enters R code in a column of the spreadsheet, and executes the code by selecting the spreadsheet range and pressing the relevant toolbar button. The user also can transfer scalar numeric and string (scalar) variables, and numeric and string matrices. Data frames are currently *not* supported, this will be added in a later version.

In this mode, Excel is used a "scratchpad and editor", and the user has to know R, and is using R through an Excel window.

2.2 Encapsulated mode

One of the examples shipped with the macro package is a spreadsheet with just one data column and a toolbar with a button labeled `Density`. Pressing this button starts R, and calculates a kernel density estimator for the data for different window widths and displays the sheet depicted in Figure 3.

It is instructive to see how the transfer of data between Excel and R is handled in this case (note that the underscore character "_" is the line continuation character in VBA):

```
Sub DoR()
Call RInterface.StartRServer
Call RInterface.PutArray("x", Range("Sheet2!A11:A1010"))
Call RInterface.REval( _
  "out<-density(x,n=100,width=1,from=-5,to=5)$x")
Call RInterface.REval( _
  "for(i in seq(.2,5,0.1))" & _
  "{out<-cbind(out,density(x,n=100,width=i,from=-5,to=5)$y)}"
  _ )
Call RInterface.GetArray("out", Range("Sheet2!G11"))
Call RInterface.StopRServer
End Sub
```

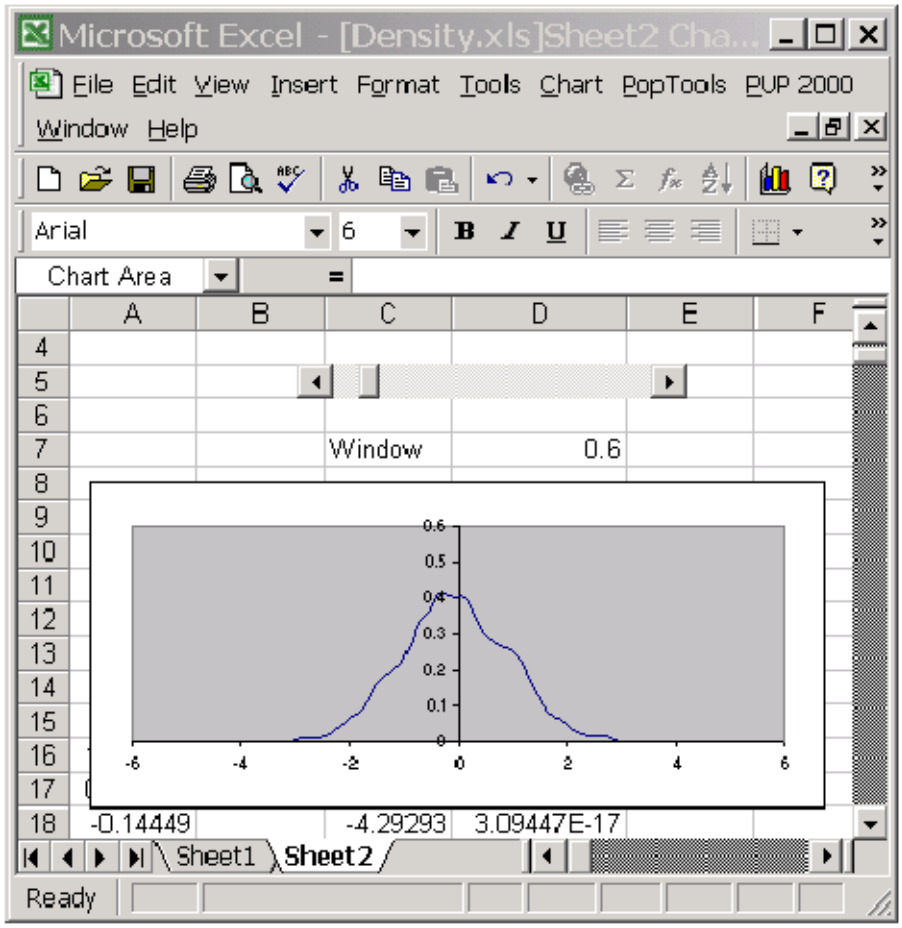

Figure 3: Animated kernel density estimator

This piece of code transfers the data to R, calculates the density estimators for different window widths in a loop, transfers these values back into a range in Excel, and then closes the connection to R. Thereafter, some Excel specific code creates a slider for controlling the window width of the KDE, and finally displays a graph with the KDE. Moving the slider automatically changes the display, so we have an animated graph of the KDE. R by itself does not offer facilities for animated graphs. Using the Tcl/Tk packages would allow to create a very similar application, but this also needs additional software installed.

In this example, we combine two software packages, R and Excel, and use easily accessible features from each one to create an even more powerful integrated application. This application can be used without any knowledge of R. Speaking more generally, a developer can develop an R application using the mechanisms described in section 2.2 and turn the developed code into an encapsulated procedure. The `Copy code` button on the toolbar in Figure 2 supports this transfer, it takes text from a region of the spreadsheet, adds a VBA-wrapper containing the strings "`Call RInterface.REval("`" and "`")`", and transfers this text to the clipboard. The developer then can switch to the VBA editor and immediately paste the VBA-code segment, which will execute the R commands, into a VBA procedure.

2.3 Automatic Recalculation

One of the most important features of spreadsheet programs is the automatic recalculation feature. Spreadsheets contain formulas in cells, and the values depend on values in other cells. Whenever a cell value is changed by the user (or changes as a consequence of some other action), the spreadsheet program automatically recalculates all formulas.

So far, our examples for connecting R and Excel did not link R to this important feature of spreadsheet programs. The main reason is that so far we have not defined any functions used in a spreadsheet, only subroutines (=procedures) which were started by pressing toolbar buttons.

Functions can be defined easily. The following code segment defines as VBA function `ncchidist` which can be called in a spreadsheet.

```
Function ncchidist(x, deg_free, noncent)
result = rexp("pchisq(" & _
         CStr(x) & "," & CStr(deg_free) & _
         "," & CStr(noncent) & ")")
ncchidist = result
End Function
```

When this function is defined, any cell formula can contain a call to the function `ncchidist`, and Excel will automatically recalculate this value when one of the input cells change.

Using this mechanism, any (scalar) R function can be turned into an Excel function by just wrapping it into a VBA function. Currently, using array functions is not fully supported, but will be implemented in a future release.

3 General considerations

Our R-Excel interface gives us 3 interface modes:
Interactive developer mode
Procedural mode of encapsulated R modules
Automatical recalculation mode of R functions
These 3 modes also address very different user groups.

Developer mode quite clearly is only of use for users with a reasonable amount of knowledge about R. This mode allows us to immediately start working on data in Excel using R. Since R is fully accessible, any packages written to extend R can be used from within the spreadsheet environment. Additionally, a developer or a power user can use Excel's user interface elements to control animated displays of results of calculations in R.

Procedural mode allows to literally put any statistics procedure written in R on an Excel button. The end user does not need to know anything about R, used this way R procedure look like a natural part of Excel, and the user might even think of "using just one more Excel routine". An expert statistician might develop routines in R and make them accessible to end users by embedding them into a spreadsheet so that the end user does not even see R.

Automatic recalculation mode in a certain sense is the most advanced way of connecting R and Excel. R is not only an addin which is called sometimes and then "laid to rest again", it becomes totally "woven into" the fundamental spreadsheet paradigm, automatic recalculation. Since spreadsheets offer some very nice facilities doing what-if scenarios, this connection allows us to integrate spreadsheet concepts and ideas and powerful statistical and mathematical calculations. Since Excel is notorious for its bad handling of some numerically delicate problems (like matrix inversion), this mode can even be used just to improve Excel as a numerical toolbox.

For a more detailed discussion of the importance of spreadsheet concepts for statistics see [1].

4 DCOM as the infrastructure

As mentioned in the first section, the implementation uses DCOM to achieve the goal of embedding R into a spreadsheet application.

COM is shorthand for *component object model*, a technology widely used on Microsoft Windows platforms for server applications to expose functionality (components) to client applications. A component is a set of functionality and data (an object) and can be as simple as e.g. the encapsulation of a simple integer value and as complex as an application like Microsoft Excel itself.

From a very simple point of view, the services COM provides and furthermore the whole architectural model itself is very similar to CORBA, an object model used mostly on Unix platforms. COM

- defines a way for a server applications to expose objects to its clients,
- defines methods to handle object life-time by simple reference-counting and

- provides standardized mechanisms for object creation and sharing.

In addition to that, COM transparently handles sharing components across process boundaries and so allows to integrate components provided by one executable file or program into another one. The previous example of Microsoft Excel embedding R as a component providing a very powerful computational engine clearly depicts this. For a client application (like Microsoft Excel) the component itself is treated just like an object provided by the application itself. Using this component then is similar to calling an internal VBA function or object.

One of the major advantages of COM itself is its wide support on the Microsoft Windows platform. Nearly every programming language, as e.g. *Visual Basic* or *C++*, scripting language as *JavaScript*, *Perl* or *Python* or even applications providing macro support as the Microsoft Office family of products (*Visual Basic for Applications*) provide support for using the functionality exposed through COM objects by a server application.

DCOM stands for *distributed COM* and extends the model described above with a very imported feature. While COM itself provides methods for performing function or method calls across process boundaries, DCOM goes one step further. DCOM makes COM objects transparently available in a network of computers. In our previous example of Microsoft Excel utilizing R as a computational component, DCOM now allows to run the component (R) and the client application (Microsoft Excel) on different machines. DCOM will take care of the necessary communications over the network to make the services exposed by the R component available to Excel.

5 The R Components

R is an interpreter for the programming language S and presents itself on the Windows platform as some kind of integrated environment. When starting R (*rgui*), the user is presented a main window containing a smaller sub-window (the console) which is used to directly enter statements in the S language and which are then executed (evaluated) by the interpreter. When plotting data, an internal graphics device is used by default, which is represented as a sub-window to the R main window, very similar to the interpreter console window.

We can identify three major components here,

- the interpreter itself (the computational engine of R),
- the console window, where the user interactively enters commands and which is used by R for output of text and
- the graphics device, used for displaying any graphics produced by R functions or scripts.

The interpreter component receives input from the console window and writes text to the console or draws graphics to the graphics device. This also is the

component, we are mostly interested in, because this component provides the computational services we want to integrate into our own (or some standard application as a spreadsheet) application.

To achieve this goal, we created a DCOM server application which uses R very similar to a mathematical library. Every single instance of the R server component manages an independent R interpreter by using R as a library providing this interpreter. By using multiple instances of the component, one also defines multiple independent R interpreters. The server now provides functions to set or retrieve data from its interpreter space and to evaluate statements in the context of its interpreter. The interface itself is defined as a COM interface and therefore allows use from many different languages, e.g. from Microsoft Visual Basic:

```
' create the R component
dim R as New StatConnector

' start the interpreter
R.Init "R"

' retrieve 100 random numbers and store them in ‘‘d’’
d = R.Evaluate ("rnorm (100)")

' shut down the interpreter
R.Close
```

In addition to the computational component (the R interpreter for the S language) we also provide COM components for the presentation part of R. There are components for a character output device, which can be used by the interpreter to output textual data. Additionally, there is a component for a graphics window, which can be used to produce high quality graphics output R provides out of the box.

Both components are modelled as Active X controls, which are COM components, that can easily be integrated into dialog windows (forms) or any other user interface components. Figure 4 depicts a very simple Visual Basic application showing the Active X control used by the R interpreter as a graphics device. The Visual Basic code used to accomplish this drawing is shown below:

```
Dim R As New StatConnector

R.Init "R"

' tell R, which graphics device to use
R.AddGraphicsDevice "dev1", Gfx1.GetGFX

' draw some lines, color ‘‘blue2’’
R.EvaluateNoReturn "plot (1:3,c(3,8,1),type=""l"",col=""blue2"")"

R.Close
```

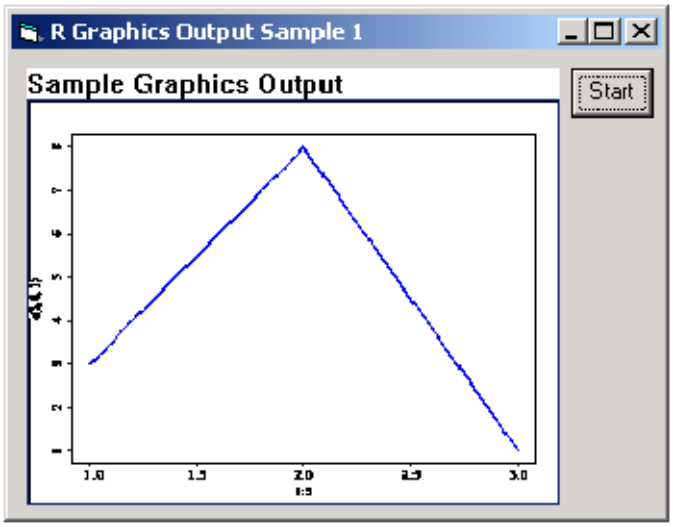

Figure 4: Custom Application With R Graphics Output

In the case of DCOM, the computational engine (`StatConnector`) will run as a DCOM server application on the server machine in the network, while the two Active X controls (graphics device and text output device) run on the client's machine. The interpreter then communicates via DCOM with the Active X controls and causes text or graphics to be displayed.

6 Application Scenarios

Finally we would like to mention some real-world application scenarios where R is embedded as the computational engine or possibly could be:

First, as presented in the beginning, Microsoft Excel, a classical spreadsheet application, is used as the front-end presented to the user, while the computational engine for complex tasks is R embedded by use of the DCOM server. Here, the user can keep interacting with a familiar user interface paradigm while still using the powerful capabilities of R as its computational engine.

In addition to a spreadsheet user interface, it is in many cases desirable to create a custom application, maybe a dialog-based user interface, for a specialized purpose, where the user is presented only the required controls and possibilities. Whereas R itself provides a user interface too crude for many end-users not familiarly with software packages of this kind, it is quite simple to create specialized applications by using development tools like Microsoft Visual Basic. Even in this case, when a user-friendly user interface is created by simply combining existing controls in a graphical editor, the computational back-end can be modelled in R and will profit from the rich set of functionality supported there. As an additional benefit, the

graphics output capabilities of R can be used by simply adding one of the previously mentioned Active X controls to a form in the application.

While it is already possible to embed R into a web application by using CGI (*common gateway interface*), there is another alternative on the Microsoft Windows platform. It is very easy to embed R into Microsoft's Internet Information Server by means of server-side scripting. By using a common technology called ASP (*active server pages*), it is possible to use R inside server-side scripts (normally using either VBScript or JavaScript) being able to use R as a statistical or computational component for large web applications, while being transparently integrated into the development environment and scripting language.

One of the major advantages of DCOM in combination with the integration of R into different types of applications is, that DCOM does not require R itself or any of the accompanying or custom libraries to be installed on the client machine. This offers the possibility of a centralized installation of R and an easy maintenance of this installation. Adding new libraries on the server automatically makes them available for all clients and even newer releases of R itself can be made available transparently without the need to update the client machines.

7 Availability

The current version of the interface packages including examples is always available from `http://cran.r-project.org/` in section "Other".

References

[1] Erich Neuwirth. Spreadsheets as tools for statistical computing and statistics education. In *Compstat, Proceedings in Computational Statistics*, 2000.