DSC 2003 Working Papers
(Draft Versions)

http://www.ci.tuwien.ac.at/Conferences/DSC-2003/

High-Level Interface Between R and Excel

Thomas Baier[*] **Erich Neuwirth**[†]

Abstract

1 Introduction

R and Microsoft Excel have a very different user interface concept. R is a programming language, whereas Excel mostly works with a direct manipulation interface. In R one works with data objects (data frames, matrices, single values) by naming them and referring to them by names in the program. The spreadsheet interface (as implemented in Excel, but also in many other spreadsheet programs like Gnumeric or OpenCalc) allows to refer to data objects by gesturing, or pointing at them. Quite often, this is understood as a GUI (graphical user interface) concept, but that is not correct. The gesturing spreadsheet interface predates graphical user interfaces. Visicalc, the very first spreadsheet program, was a character mode application and already implemented this approach.

Combining R and Excel brings together these two very different user interface worlds, and therefore, besides technical implementation issues, one also has to be very careful when combining radically different user interaction concepts. More information how the spreadsheet interface is very useful for statistics can be found in [3]

We implemented 2 different philosophies. In one package, R is embedded as an extension of Excel. The user stays within the spreadsheet interaction model and gets additional functionality. One might say this is a statistically enriched version of Excel.

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

In the second package, Excel is made accessible from within R. The user enters R code, and some of the R commands start Excel and allow to enter or edit data in Excel, or some results of statistical computations in R into spreadsheet ranges. One might say that Excel becomes accessible as a scratchpad for R data and R output.

Let us study these two approaches in more detail.

2 Excel as the host

Extending the functionality of Excel usually is done in two different ways. One can either add new menus and menu items offering operations on data in the spreadsheets, or one can define new functions which can be used in cells formulas.

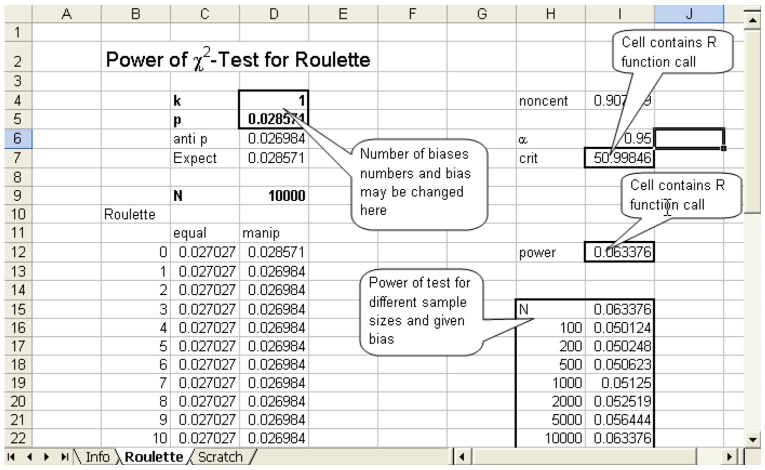
The important difference is that menu operations produce static output, whereas in-cell function will update automatically when the input cells for a formula change. For every spreadsheet, Excel internally builds the dependency tree and whenever any cell is changed, all cells depending on that cell will automatically be recalculated. Here is an example of the most simple form of using R functions in Excel (through the RExcel addon package):

```
RApply("pchisq",30,1)
```

This formula computes the probability of the chisquare distribution for the given values.

Since in our package R is fully integrated into Excel, the second and the third argument in this call may be references to cells. Using this mechanism, we can calculate the power of a statistical test in an Excel sheet. Excel has some statistical distribution functions including the chisquare distribution in the Data Analysis Toolkit, but only the central version. Since R has the noncentral distribution function the advantage of getting R to work within Excel is immediate: we can calculate the power of chisquare tests in Excel.

The example supplied with our RExcel package calculates the power function of a chisquare test for a Roulette wheel with unequal probabilities when testing for equal probabilities.



	A	B	C	D	E	F	G	H	I	J
1										
2		Power of χ^2-Test for Roulette								
3										
4			**k**	**1**				noncent	0.907[illegible]9	
5			**p**	**0.028571**						
6			anti p	0.026984				α	0.95	
7			Expect	0.028571				crit	50.99846	
8										
9			**N**	**10000**						
10		Roulette								
11			equal	manip						
12		0	0.027027	0.028571				power	0.063376	
13		1	0.027027	0.026984						
14		2	0.027027	0.026984						
15		3	0.027027	0.026984				N	0.063376	
16		4	0.027027	0.026984				100	0.050124	
17		5	0.027027	0.026984				200	0.050248	
18		6	0.027027	0.026984				500	0.050623	
19		7	0.027027	0.026984				1000	0.05125	
20		8	0.027027	0.026984				2000	0.052519	
21		9	0.027027	0.026984				5000	0.056444	
22		10	0.027027	0.026984				10000	0.063376	

Cell contains R function call
Number of biases numbers and bias may be changed here
Cell contains R function call
Power of test for different sample sizes and given bias
Info
Roulette
Scratch

Excel gives a very convenient interactive way of doing what-if analyses, for example by changing parameters of a statistical model.

`RApply` turns any R functions into an Excel function. Sometimes we might want to hide the R functions completely from the user. We can do this by putting a VBA wrapper around the call to R.

In our case, we can define

```
Function ncchidist(x, deg_free, noncent)
ncchidist=RApply("pchisq",x, deg_free, noncent)
End Function

Function ncchiinv(prob, deg_free, noncent)
ncchiinv=RApply("qchisq", prob, deg_free, noncent)
End Function
```

These two functions can then be used like any other Excel function, and using this mechanism we can make any set of R functions (including ones defined in user code) accessible in Excel transparently.

The software connection between R and Excel is implemented by the making R a COM server. The COM server has been described in [2]

This can be done in two different ways, which will be discussed later. At first, we will have a look at the interface.

The complete interface for using R in spreadsheet functions called directly in spreadsheet cells in Excel is implemented with the following functions:

Name	Description
`RApply`	apply an R function to the given arguments
`REval`	evaluate an R expression given as a string
`RVarSet`	Assign an R expression given as a string to an R variable
`RPut`	Assign a value from an Excel range to an R variable
`RStrPut`	Assign a value from an Excel range to an R variable as a string
`RProc`	Execute some R commands

Let us discuss these new Excel functions briefly:
`RApply` and `REval` evaluate R function calls and put the result in a cell. The difference is that `RApply` expects the function as the first argument and the arguments for the function as the remaining arguments whereas `REval` needs a string which is a complete R function call including the arguments as its argument. Typical uses are `RApply("sin",1)` and `RExp("sin(1)")`. In both cases, all the arguments can be references to other spreadsheet cells.

Since `RApply` performs a function call with a function as the first argument to `RApply` and the arguments to this function as the remaining arguments, and since R is a full functional language with functions as first class objects, one also can make function calls like `RApply("function(x)x*x",2)` and again, the arguments for `RApply` can be cell references to other spreadsheet cells. This we, we can define R function on the spot within a spreadsheet, and we even do not need to name them.

For functions with more code, however, being able to define named function is quite useful. `RProc` takes a spreadsheet columnar spreadsheet range and executes the text in then range as R code. Writing a function definition in the usual way

```
myfun<-function(x){
x*x
}
```

and calling `RProc` with this range as argument will define function `myfun` in R. Then, `RApply` can use `myfun` as its fist argument.

This additional mechanism introduces an new problem: recalculation order. For every spreadsheet, R has a dependency table so when any cell in changed, it recalculates all the cell depending on this cell. If in some cell in our spreadsheet we use a function defined using `RProc` in another cell, Excel does not know that the function definition has to be evaluated before the function is evaluated. We will discuss this problems immediately after we have looked at the other interface functions.

`RVarSet` has two arguments, a variable name and a string expression. The string expression must be a valid R expression and the value of this expression is assigned to the variable. The string expression can be constructed by combining contents of spreadsheet cells with Excel's string functions. `RVarSet(A1&"*"&A2)` would use R to multiply the contents of cell `A1` and `A2` in a spreadsheet.

`RPut` and `RStrPut` Assign a value from an Excel range to an R variable. RPut will assign a numeric value if all the cells referenced in the second argument contain numeric values, `RStrPut` will enforce that the assigned values in R are string values. If the range consist of only the assigned object will be a scalar, otherwise the object will be either a numeric or a string matrix.

Now, let us discuss the dependency and recalculation order issue raised when discussing `RProc`. We introduce what we might call *artificial dependencies*. `REval`, `RProc`, `RVarSet`, `RPut`, and `RStrPut` all have a fixed number of arguments for normal use. They will take additional arguments. These arguments will be evaluated within Excel before the operation itself is performed. If the first argument of `REval` is a function call for an R function defined in some other cell(s) of the spreadsheet (possibly using `RProc`), a second argument to `REval` referencing the cell where the definition is executed (this is the cell with the spreadsheet formula containing `RProc`, *not* the cell(s) with the code defining the function), will enforce that the definition is performed before the function is evaluated. This is especially important when the function definition is changed. In that case, without indicating the dependency, the evaluation call might use an obsolete definition of a function. Things are a little bit more complicated for `RApply`. `RApply` takes a variable number of arguments. Therefore, we need an indicator to separate real arguments (the ones used in the R function call) from arguments used only to indicate dependencies. If an argument to `RApply` is the string `"depends"`, then all arguments to `RApply` will not be used in the R function call created by `RApply`.

Currently, our implementation runs reasonably well for the R (D)COM server described in [**?**]. This way, R is imbedded invisibly into a spreadsheet application, there is no GUI or command line interface to the R instance doing the computational work for Excel. This R instance can only be accessed from within Excel.

A second implementation make it possible to access an instance of R that such that Excel and a command line or GUI version of R share name, data, and code space, providing access to the R command line at the same time as to the spreadsheet interface. This allows to use the strengths of both office applications and R side by side on the Windows platform. Thomas Baier's paper [1] gives the details of this implementations.

Instead of using spreadsheet functions as the main interface between R and Excel we also can use VBA (the programming language embedded in Exel). This interface (it is the first one in our project of combining R and Excel is described in [2]. The interface consist of 5 VBA procedures:

Name	**Description**
RInterface.StartRServer	starts a new R server instance
RInterface.StopRServer	stops the current R server instance
RInterface.RRun	executes a line of R code
RInterface.PutArray	transfers an array from Excel to R
RInterface.GetArray	transfers an array from R to Excel

Using these VBA procedures it is possible to write VBA programs with full access to R. There is a very important difference between embedding R into Excel with spreadsheet functions and with VBA procedures: when the function interface is used the execution of R functions becomes part of Excel's automatic recalculation. When the VBA-procedural interface is used, only the VBA programs (possibly triggered by menu items and buttons) start the execution of R code. In the standard case this means that R code is only executed when the user presses a button or selects a menu item.

Here is a short example

```
Sub Demo()
Call RInterface.StartRServer
Call RInterface.RRun("z<-rnorm(60)")
Call RInterface.RRun("dim(z)<-c(10,6)")
Call RInterface.GetArray("z", Range("Sheet1!A19"))
Call RInterface.StopRServer
End Sub
```

This procedure creates a vector of 60 random numbers, changes the vector into a 10x6 matrix and then transfers the matrix to an Excel range. It can be attached to a button or menu item, and then the user can trigger this operation, effectively creating a matrix of random numbers produced by R in Excel.

3 R as the host

The other way of embedding is using R as the COM client and, say, Excel as the COM server. In this case, "R is in control", meaning that the user mainly interacts with the R command line. A relatively straightforward use would be calling Excel from R to get some data which are already available as an Excel spreadsheet, then doing complex statistical analyses, and finally putting some reports (possibly including graphics) back into Excel. In theory, this can also be done using our previously described case where R acts as COM server. The main difference is that in the first framework the user works with Excel enhanced by R, and in the second case works with R with the additional possibility of accessing Excel from within R.

Excel, in this case, just is a representative for any application exposing it's functionality as a COM server. Especially, this applies to all applications in Microsoft's Office family of products. In our discussion, R is responsible for the computational functionality, while the Office suite is used for user input and output/presentation.

Using R'c COM client library `rcom`, one can access any COM server's functions as long as they are provided in a form conforming to *OLE automation*. In addition to this low-level mechanism, we are providing an R package especially tailored to the basic requirements for interaction with Microsoft Excel. The goal of this effort is to simplify the most common applications, while still providing full access to all functionality.

We may enhance R's GUI interface by adding additional menu items. These can used for easy integration of foreign functionality into R's command-line driven GUI. As an example, basic integration of part of the Microsoft Office suite is shown.

Excel functionality can be implemented in R rather smoothly by adding menu functions for transferring data. For this reason, only the integration of Microsoft Excel is discussed in more detail for this part by utilizing the `rexcel` high level access package.

3.1 Accessing Excel, PowerPoint and Word

Even when using R as the primary workbench for data analysis, in the Windows world, presentations of output are expected to be done using Microsoft's Office suite of programs, as well as input data is many times available in e.g. Microsoft Excel file formats.

Sometimes it can be convenient to drive PowerPoint or Word from the R command line or get data from an Excel spreadsheet. As all Office applications expose their functionality as an object model via COM, the `rcom` package can be used for access. The following code shows some R functions to start up and show Excel, PowerPoint and Word:

```
# to be reworked. doesn't really work in generic case!
office.get<-function(app,handle) {
  if(!identical(handle,NULL)) return(office.show(handle));
  handle<<-com.object.get(app);
  if(identical(handle,NULL)) handle<<-com.object.create(app);
  return(office.show(handle));
}
office.show<-function(handle) {
  if(!identical(handle,NULL)) {
    com.property.set(handle,"Visible",TRUE);
  }
}
```

To provide some basic integration into R's user interface, an additional package is in development, which allows to add new menu items into the R workbench. This package—`macros`—is used to associate this function with user interface elements. Adding a new pull-down menu `Office` with menu items for starting Excel, PowerPoint and Word are achieved by the following code:

```
menu.create("Office")
menu.create.item("Start Excel","Office",
          "office.start(\"Excel.Application\",excel.handle)")
menu.create.item("Start PowerPoint","Office",
          "office.start(\"PowerPoint.Application\",excel.handle)")
menu.create.item("Start Word","Office",
          "office.start(\"Word.Application\",excel.handle)")
```

As well as starting these applications and manipulating properties it is easy as well to call functionality and transfer data between R and Office.

3.2 Controlling Excel: `rexcel`

The object model of many applications is very complex and requires the user to extensively study help files and manuals. For many cases, only a small subset of all functionality is required.

The 90:10 rule also applies to the requirements of users and their applications: 90% of your needs are covered by only 10% of the functionality.

For our previous work we have been focusing on Microsoft Excel as a very useful medium for data entry and output. Therefore, we decided to provide an easy-to-use interface to a small subset of functions in Excel which can easily be integrated into the R GUI and R programs.

What we have been focusing on is providing access to

- getting access to Excel itself (either a current instance or starting a new window)
- exchanging Excel's current selection with R (reading the selection into an R variable or writing a variable to the current selection)
- exchange any rectangular area in the spreadsheet with R

In the case of data transfer, the functions are trimmed to either return a value (e.g. a data vector or matrix) or to provide access to the Excel COM object itself.

The following functions have been implemented:

Name	**Description**
`excel.get`, `excel.create`	return the current instance of Microsoft Excel or create a new instance
`excel.show`	show the Excel window
`excel.selection.value`	get selection value
`excel.selection.value.set`	set selection value
`excel.selection.get`	get selection object
`excel.cell.value`	get cell value
`excel.cell.value.set`	set cell value
`excel.cell.get`	get cell object
`excel.range.value`	get range value
`excel.range.value.set`	set range value
`excel.range.get`	get range object

Using this functions, you can easily transfer data between R and Excel. The following example shows how to store the specified cell range into R's variable `rng1`.

```
# get range "A1" to "D7" and store the matrix into rng1
rng1<-excel.range.value("A1:D7")

# add 5 to every element and store back
excel.range.value.set("A1:D7",rng1+5)
```

In the above example, the currently active instance of Excel is used automatically. If none is runnig, a new Excel is created (see documentation for `excel.get()` and `excel.range.value()` for more information).

Using the COM client package, we have shown, that R can control the Microsoft's Office suite of applications. For easy access to Excel's basic input and output facilities we can use the additional package `rexcel`, for the not so common operations, you have to fall back to `rcom` and directly access the COM interfaces provided by Excel. The COM level is also the interface currently supported for the other applications of the Office suite.

As `rexcel` builds on the low-level functionality provided by `rcom` and is completely written in R itself, one can easily extend the package or seamlessly itegrate similar functionality for other applications.

In the next section, we will show an example giving a better idea of the wide range of applications of R's office integration.

3.3 Excel → R → PowerPoint

Office integration not only means, that R can get data from Microsoft Excel or pop up another application. In fact, R can completely control any of the Office applications—at least as far as Offices allows this.

In a very simple example we will show how to use Excel as a medium for user input and PowerPoint to show the results of the computations done in R.

Our simple application will use the χ^2-test to compute the probability of finding out whether a given roulette table has been manipulated. We will use R to do the computations and show the correlation between the number of observations and the probability of discovery in a PowerPoint presentation.

Here, R will read the selection from Excel. The selection shall be a matrix with two columns and 37 rows, the first column representing probabilities of the numbers 0 to 36 for a fair roulette table. The second column represents the probabilities for the manipulated roulette.

We will use some simple functions for the computations:

```
noncent<-function(n,equal,manipulated)
{
  return(n * sum(((manipulated - equal) ** 2) / equal));
}
power<-function(n,equal,manipulated,dof,alpha)
{
  nc<-noncent(n,equal,manipulated);
  crit<-qchisq(alpha,dof,0);
  return(1 - pchisq(crit,dof,nc));
}
do.test<-function(alpha,report)
{
  selection<-excel.selection.value()
  observations<-c(100,200,500,1000,2000,5000,10000,
                  20000,50000,100000,200000,500000,1000000);
  probabilities<-power(observations,selection[,1],selection[,2],
                       36,alpha);
  eval(call(report,observations,probabilities));
}
```

The front-end for the application code is `do.test`. This function takes the α value and a reporting function as its parameters. As a reporting function, one can e.g. use the `plot()` function. Adding the functionality to the main menu for different α values:

```
menu.create("ChiSquare-Test");
menu.create.item("alpha = 0.75","ChiSquare-Test",
                 "do.test(0.75,\"plot\")");
menu.create.item("alpha = 0.80","ChiSquare-Test",
                 "do.test(0.80,\"plot\")");
menu.create.item("alpha = 0.85","ChiSquare-Test",
                 "do.test(0.85,\"plot\")");
menu.create.item("alpha = 0.90","ChiSquare-Test",
                 "do.test(0.90,\"plot\")");
menu.create.item("alpha = 0.95","ChiSquare-Test",
                 "do.test(0.95,\"plot\")");
menu.create.item("alpha = 0.97","ChiSquare-Test",
                 "do.test(0.97,\"plot\")");
```

Instead of using `plot()` for presenting the results, we will use Microsoft PowerPoint as an output device. A report function compatible with the above parameter list is shown below:

```
ppoint.report<-function(n,probs)
{
  # start PowerPoint and create an presentation with a single slide
  ppt<-com.object.create("PowerPoint.Application")
  pres<-com.invoke(com.property.get(ppt,"Presentations"),"Add")
  slides<-com.property.get(pres,"Slides")
  # type 12 is empty slide, 8 is chart, 2 is title+enumeration
  slide<-com.invoke(slides,"Add",2,1)

  # set the title of the slide
  title<-com.property.get(com.property.get(slide,"Shapes"),"Title")
  rng<-com.property.get(com.property.get(title,"TextFrame"),
                        "TextRange")
  com.property.set(rng,"Text","Roulette: Chi Square-Test")

  # add analysis results to slide
  enumshape<-com.invoke(com.property.get(slide,"Shapes"),"Item",2)
  rng<-com.property.get(com.property.get(enumshape,"TextFrame"),
                        "TextRange")
  txt<-paste(n,rep("observations\tp =",length(probs)),probs,
             collapse="\r");
  com.property.set(rng,"Text",txt);
  com.property.set(com.property.get(rng,"Font"),"Size",16)
  # show the results
  com.property.set(ppt,"Visible",TRUE)
  com.invoke(com.property.get(pres,"SlideShowSettings"),"Run")
}
```

And of course, this reporting function has to be specified when creating the menu items, e.g.

```
...
menu.create.item("alpha = 0.75","ChiSquare-Test",
                 "do.test(0.75,\"ppoint.report\")");
...
```

This provides a very simple data reporting facility. The user chooses a menu item in the R application window, R gets the data from Excel, computes the results and puts the report into PowerPoint. After the results have been shown, you can simply save the resulting presentation to disk for later use.

4 Comparing the philosophies

In addition to embedding an invisible R into a spreadsheet application, it is also possible to run R in parallel to the spreadsheet, providing access to the R command line at the same time as to the spreadsheet interface. This allows to use the strenghts of both office applications and R side by side on the Windows platform.

References

[1] Thomas Baier. Embedding R in standard software, and the other way round. In Kurt Hornik and Friedrich Leisch, editors, *DSC 2003 Proceedings of the 3rd International Workshop on Distributed Statistical Computing*, http://www.ci.tuwien.ac.at/Conferences/DSC-2003/, 2003.

[2] Thomas Baier and Erich Neuwirth. Embedding R in standard software, and the other way round. In Kurt Hornik and Friedrich Leisch, editors, *DSC 2001 Proceedings of the 2nd International Workshop on Distributed Statistical Computing*, http://www.ci.tuwien.ac.at/Conferences/DSC-2001/, 2001. ISSN: 1609-395X.

[3] Erich Neuwirth. Spreadsheets as tools for statistical computing and statistics education. In Jelke G. Bethlehem and Peter G. M. Van Der Heijden, editors, Compstat, Proceedings in Computational Statistics, 2000.