*Proceedings of the 3rd International Workshop*
*on Distributed Statistical Computing (DSC 2003)*
*March 20–22, Vienna, Austria     ISSN 1609-395X*
*Kurt Hornik, Friedrich Leisch & Achim Zeileis (eds.)*
*http://www.ci.tuwien.ac.at/Conferences/DSC-2003/*

# High-Level Interface Between R and Excel

**Thomas Baier**     **Erich Neuwirth**

**Abstract**

Spreadsheet programs in general and Microsoft Excel in particular implement a very special paradigm of computing: automatic creation of dependency structures for calculation, and automatic recalculation when input values change. This is rather different from the usual model of batch processing underlying most statistics software, e.g. R.

Since some time now, a COM server implementation for R has been available. As a more recent development, R can also act as COM client, using Excel as a server. So we can either think of R as an extension of Excel, or of Excel as a utility program (and data editor) for R.

We will discuss the differences between these 2 approaches, and their relative merits under different circumstances. This will be done by showing examples of applications implementing these rather different models for statistical computation.

Emphasis will be more on the philosophy and style of statistical analysis than on technical implementation details.

## 1 Introduction

R (Ihaka and Gentleman, 1996) and Microsoft® Excel (Microsoft Corporation (1999)) have a very different user interface concept. For R most interaction happens through the high-level R 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 (GNOME Foundation, 2003) or OpenCalc (OpenCalc Technical Team, 2003) allows to refer to data objects by gesturing, or pointing at them. Quite often, this is understood as a GUI (graphical user interface) concept, which, however, is not correct. The gesturing spreadsheet interface predates graphical user interfaces. Visicalc (Bricklin, 1979), 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, apart from technical implementation issues, one also has to be very careful when combining radically different user interaction concepts. More information indicating how the spreadsheet interface is very useful for statistics can be found in Neuwirth (2000), and more generally in science and engineering applications can be found in Filby (1997), especially Neuwirth (1997).

We implemented 2 different philosophies. In the first approach, R is embedded as an extension of Excel. The user stays within the spreadsheet interaction model and gets additional functionality provided by R. One might say this is a statistically enriched version of Excel. This will be discussed in section 2.

In the second approach, Excel functionality 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. Alternatively, results of statistical computations in R are written into spreadsheet ranges. One might say that Excel becomes accessible as a scratch-pad for R data and R output. This approach is discussed in section 3.

## 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 cell formulas.

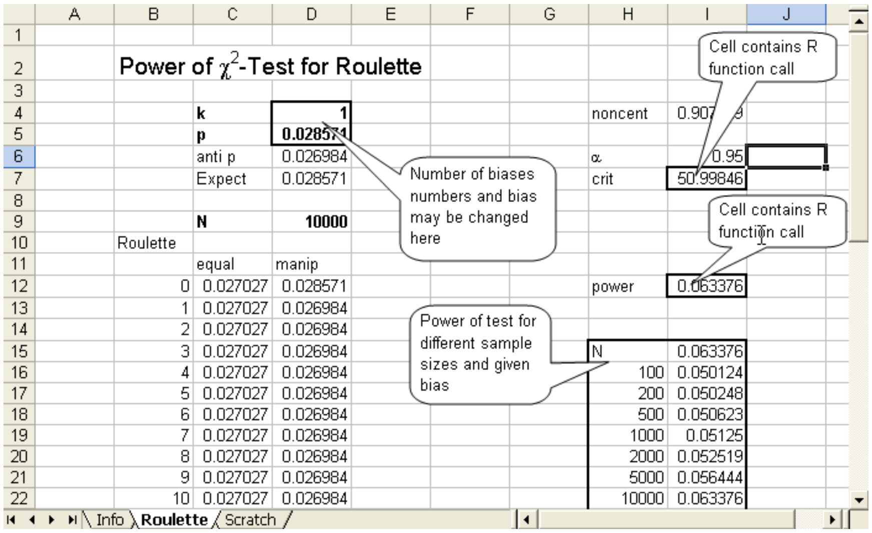
The important difference is that menu operations produce static output, whereas in-cell functions will update automatically when the input cells for a formula change. For every spreadsheet, Excel internally builds the dependency tree and whenever a cell is changed, all cells depending on this cell will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

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

This formula computes the probability function of the chi-squared distribution for the given values.

In terms of Excel, R (or more specific: R's services) is a first-class object in Excel. This also allows the arguments in a call to `RApply` to be references to cells. Using this mechanism one can calculate the power of a statistical test in an Excel sheet. Excel has some statistical distribution functions including the chi-squared distribution in the *Data Analysis Toolkit*, but only the central version. Since R has the non-central distribution function the advantage of getting R to work within Excel is immediate: one can calculate the power of chi-squared tests in Excel.

The example supplied with our RExcel add-in calculates the power function of a chi-squared test for a Roulette wheel with unequal probabilities when testing for equal probabilities.



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

|  | A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 |  |  |  |  |  |  |  |  |  |  |
| 2 |  | Power of $\chi^2$-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 |  |

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 function 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 (*V*isual *B*asic for *A*pplications) wrapper around the call to R. VBA is an abbreviation for the macro language used in Excel (and in general in all Microsoft Office applications). See Microsoft Corporation (2002) for more information.

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) transparently accessible in Excel.

The software connection between R and Excel is implemented by making R a COM server. The COM server has been described in Baier and Neuwirth (2001).

There are two different implementations of R as a COM server available. One of them provides R's functionality as an invisible service application (comparable to e.g. a database), whereas the other implementation allows user interaction with R's GUI interface at the same time as R's computational engine is used by Excel. More details on this will be discussed later.

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 |

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 REval("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 fully 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 is how one can define an R function on the spot within a spreadsheet, and this can even be an anonymous function.

For functions with more code, however, being able to define named function is quite useful. RProc takes a columnar spreadsheet range and executes the text in the 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.

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,B1&"*"&B2) would use R to multiply the contents of cells B1 and B2 in a spreadsheet and assign the result to the variable whose name is given in cell A1. Note that in this example VBA's string concatenation operator & is used to build the string argument.

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 consists of only one cell the assigned object it will be a scalar. Otherwise the object will either be a numeric or a string matrix.

Allowing named functions defined in a spreadsheet cell to be used in the calculations in other cells introduces new problems: dependencies and recalculation order. For every spreadsheet, Excel has a dependency table so when any cell is changed, it recalculates all the cells 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.

To solve this problem 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 no arguments to `RApply` will be used in the R function call created by `RApply`.

Currently, our implementation runs reasonably well for the R (D)COM server described in Baier and Neuwirth (2001). This way, R is embedded invisibly into a spreadsheet application. There is no GUI or command line interface to the R instance doing the computational work for Excel. The R instance can only be accessed from within Excel.

A second implementation makes it possible to access an instance of R in a way that Excel and a command line or GUI version of R share name, data, and code space. Access to the R command line is provided 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. See Baier (2003) for details of this implementations.

Instead of using spreadsheet functions as the main interface between R and Excel we can also use VBA (the programming language embedded in Excel). This interface (it is the first one in our project of combining R and Excel) is described in Baier and Neuwirth (2001), and consists 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, transforms 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 obtain data 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 its functionality as a COM server. In particular, 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's COM client package **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 the R package **ROffice.Excel**. The purpose of this package is to allow easy interaction between R and Microsoft Excel. It will simplify the most common applications by providing high-level functionality especially designed as an interface to Excel, while still providing full access to all functionality.

We may enhance R's programmable GUI available under Windows by adding additional menu items. These can be used for easy integration of foreign functionality into R's more text-based interface. 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 **ROffice.Excel** 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. At the same time, many times input data for computations is available in e.g. Microsoft Excel file format.

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 PowerPoint and Word:

```
.ppointHandle <- local({
  .pp.loc <- character(0)
  function(new) {
    if(!missing(new)) {
      .pp.loc <<- NULL
    } else {
      .pp.loc
    }
  }
})
ppointCreate <- function(storeglobal = TRUE) {
  pp <- comCreateObject("PowerPoint.Application")
  if(storeglobal) {
    .ppointHandle(pp)
  }
  return(pp)
}
ppointGet <- function(createonerror=TRUE,
                      storeglobal=TRUE,useexisting=TRUE) {
  if(useexisting && !identical(.ppointHandle(),NULL)) {
    return(.ppointHandle())
  }
  pp <- comGetObject("PowerPoint.Application")
  if(createonerror && identical(pp,NULL)) {
    pp <- ppointCreate(storeglobal = FALSE)
  }
  if(storeglobal) {
    .ppointHandle(pp)
  }
  return(pp)
}
ppointShow <- function(visible=TRUE,pp=ppointGet()) {
  comSetProperty(pp,"Visible",visible)
}
.wordHandle <- local({
  .word.loc <- character(0)
  function(new) {
    if(!missing(new)) {
      .word.loc <<- NULL
    } else {
      .word.loc
    }
  }
})
wordCreate <- function(storeglobal = TRUE) {
  w <- comCreateObject("Word.Application")
  if(storeglobal) {
    .wordHandle(w)
  }
  return(w)
}
wordGet <- function(createonerror=TRUE,
                    storeglobal=TRUE,useexisting=TRUE) {
  if(useexisting && !identical(.wordHandle(),NULL)) {
    return(.wordHandle())
  }
  w <- comGetObject("Word.Application")
  if(createonerror && identical(w,NULL)) {
    w <- ppointCreate(storeglobal = FALSE)
  }
  if(storeglobal) {
    .wordHandle(w)
  }
  return(pp)
}
wordShow <- function(visible=TRUE,w=wordGet()) {
  comSetProperty(w,"Visible",visible)
}
```

To provide some basic integration into R's user interface one can use the functions `winMenuAdd()` and `winMenuAddItem()` to add new menu items to R's GUI.

Adding a new pull-down menu `Office` with menu items for starting Excel, PowerPoint and Word is achieved by the following code:

```
winMenuAdd("Office")
winMenuAddItem("Office","Start Excel","excelShow()")
winMenuAddItem("Office","Start PowerPoint","ppointShow()")
winMenuAddItem("Office","Start Word","wordShow()")
```

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: ROffice.Excel

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** |
|---|---|
| `excelGet`, `excelCreate` | return the current instance of Microsoft Excel or create a new instance |
| `excelShow` | show the Excel window |
| `excelGetSelectionValue` | get selection value |
| `excelSetSelectionValue` | set selection value |
| `excelGetSelection` | get selection object |
| `excelGetCellValue` | get cell value |
| `excelSetCellValue` | set cell value |
| `excelGetCell` | get cell object |
| `excelGetRangeValue` | get range value |
| `excelSetRangeValue` | set range value |
| `excelGetRange` | get range object |

Using these 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 <- excelGetRangeValue("A1:D7")

## add 5 to every element and store back
excelSetRangeValue("A1:D7",rng1+5)
```

In the above example, the currently active instance of Excel is used automatically. If none is running, Excel is started and the new Excel object is returned (see documentation for `excelGet()` and `excelGetRangeValue()` for more information).

Using the COM client package R can control the Microsoft's Office suite of applications. For easy access to Excel's basic input and output facilities the additional package **ROffice.Excel** can be used. For the not so common operations, one hsa to rely on **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 **ROffice.Excel** builds on the low-level functionality provided by **rcom** and is completely written in R itself, one can easily extend the package or seamlessly integrate similar functionality for other applications.

In the next section, we will provide 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 Office 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.

The simple application will use the $\chi^2$ goodness-of-fit 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 slide.

Here, R will read the selection from Excel. For the example the selection is 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. The helper functions are designed to handle different cases e.g. various numbers of numbers with equal probabilities and manipulated numbers.

```
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))
}
doTest <- function(alpha,report)
{
  selection <- excelGetSelectionValue()
  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 `doTest`. This function takes the $\alpha$ value and a reporting function as its arguments. As a reporting function, one can e.g., use the `plot()` function. Now the functionality is added to R's main menu in the R GUI for different $\alpha$ values:

```
winMenuAdd("Chi-Squared Test")
winMenuAddItem("Chi-Squared Test","alpha = 0.75",
               "doTest(0.75,\"plot\")")
winMenuAddItem("Chi-Squared Test","alpha = 0.80",
               "doTest(0.80,\"plot\")")
winMenuAddItem("Chi-Squared Test","alpha = 0.85",
               "doTest(0.85,\"plot\")")
winMenuAddItem("Chi-Squared Test","alpha = 0.90",
               "doTest(0.90,\"plot\")")
winMenuAddItem("Chi-Squared Test","alpha = 0.95",
               "doTest(0.95,\"plot\")")
winMenuAddItem("Chi-Squared Test","alpha = 0.97",
               "doTest(0.97,\"plot\")")
```

`doTest()` is passed a *reporting* function presenting the results. In addition to the simple `plot()` function used before, one can also use some more sophisticated output device. The following example will use Microsoft PowerPoint as an output device. A report function compatible with the above argument list is shown below:

```
ppointReport <- function(n,probs)
{
  # start PowerPoint and create an presentation with a single slide
  ppt <- comCreateObject("PowerPoint.Application")
  pres <- comInvoke(comGetProperty(ppt,"Presentations"),"Add")
  slides <- comGetProperty(pres,"Slides")
  # type 12 is empty slide, 8 is chart, 2 is title+enumeration
  slide <- comInvoke(slides,"Add",2,1)

  # set the title of the slide
  title <- comGetProperty(comGetProperty(slide,"Shapes"),"Title")
  rng <- comGetProperty(comGetProperty(title,"TextFrame"),
                        "TextRange")
  comSetProperty(rng,"Text","Roulette: Chi Squared-Test")

  # add analysis results to slide
  enumshape <- comInvoke(comGetProperty(slide,"Shapes"),"Item",2)
  rng <- comGetProperty(comGetProperty(enumshape,"TextFrame"),
                        "TextRange")
  txt <- paste(n,rep("observations\tp =",length(probs)),probs,
               collapse="\r")
  comSetProperty(rng,"Text",txt)
  comSetProperty(comGetProperty(rng,"Font"),"Size",16)

  # show the results
  comSetProperty(ppt,"Visible",TRUE)
  comInvoke(comGetProperty(pres,"SlideShowSettings"),"Run")
}
```

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

```
...
winMenuAddItem("Chi-Squared Test","alpha = 0.75",
               "doTest(0.75,\"ppointReport\")")
...
```

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 Conclusion and outlook

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 strengths of both office applications and R side by side on the Windows platform.

Currently packages for accessing office applications are still in development. **ROffice.Excel** is available as a first implementation from the R COM home page http://sunsite.univie.ac.at/rcom. There is also a mailing list available for discussions on the current and future implementations.

# References

Thomas Baier. R: Windows component services. integrating R and Excel on the COM layer. In Kurt Hornik and Friedrich Leisch, editors, *DSC 2003 Proceedings of the 3rd International Workshop on Distributed Statistical Computing*, 2003. URL http://www.ci.tuwien.ac.at/Conferences/DSC-2003/. ISSN: 1609-395X.

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*, 2001. URL http://www.ci.tuwien.ac.at/Conferences/DSC-2001/. ISSN: 1609-395X.

Dan Bricklin. Visicalc material, 1979. URL http://www.bricklin.com/visicalc.htm.

Gordon Filby, editor. *Spreadsheets in Science and Engineering*. Springer Verlag, 1997. ISBN: 3-540-61253-X.

GNOME Foundation. Gnumeric documentation, 2003. URL http://www.gnome.org/projects/gnumeric.

Ross Ihaka and Robert Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996.

Microsoft Corporation. *Microsoft Excel*. Microsoft Corporation, 1999.

Microsoft Corporation. Online resources. In *Microsoft Visual Basic for Applications*. Microsoft Corporation, 2002. URL http://msdn.microsoft.com/vba/technical/online.asp.

Erich Neuwirth. Spreadsheets as tools in mathematical modeling and numerical mathematics. In Gordon Filby, editor, *Spreadsheets in Science and Engineering*, pages 87–114. Springer Verlag, 1997. ISBN: 3-540-61253-X.

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.

OpenCalc Technical Team. OpenCalc documentation, 2003. URL http://sc.openoffice.org.

## Affiliation

Thomas Baier
Department of Statistics
Vienna University of Technology
E-mail: baier@ci.tuwien.ac.at

Erich Neuwirth
Department of Statistics and Decision Support Systems
University of Vienna
E-mail: erich.neuwirth@univie.ac.at