DCOM, R and Excel

Duncan Temple Lang
Department of Statistics
UC Davis

Outline

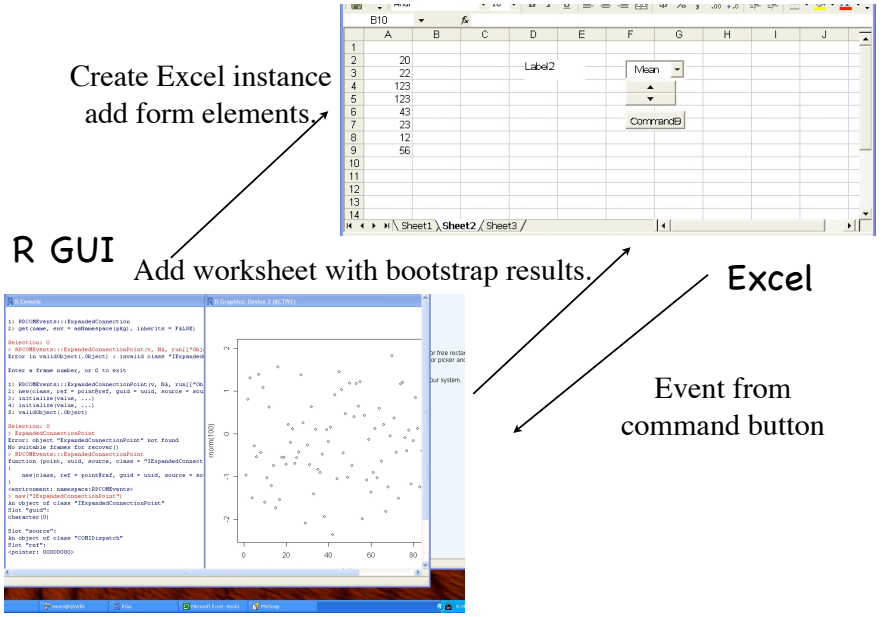
- An example
- The DCOM model
- RDCOM - facilities in R.
- Excel model.
- Events
- Reflection in R.
- Mention R-DCOM Servers

Example

- Bootstrap example that we looked at for CGI but this time in Excel.
- Take inputs from the user for the statistic to be computed for each bootstrap sample, and the number of bootstrap samples to create.
- Outputs will be
  - Numerical summary of the bootstrap distribution.
  - Density plot of the distribution
  - All the samples and the associated statistic.
- Displayed on a new worksheet.

General comments

- Excel is very popular tool, familiar to many, and relatively easy to use.
- Covenient workflow model for arranging computations in visual manner.
- However, computational engine is poor for statistics incomplete and poor accuracy and precision in statistical methods (See McCullagh, The American Statistician)
- Programming languages not very good
  Visual Basic - poor language
  C++ - too low level when we have R.
- Very beneficial to merge R's functionality and language with Excel's interface and visual programming metaphor.



DCOM

- R and Excel are two separate applications, yet we need to be able to control Excel from within R.
  And we need Excel to be able to signal to R to do something.
- So we need some sort of inter-application communication.
- Need to be able to pass data from R to Excel and to be able to call R functions from within Excel.
- This is bi-directional communication.
  Exchanging data files won't work.

- DCOM stands for Distributed Component Object Model.
- R and Excel act as Components of a bigger application that we are building.
- Both R and Excel provide Objects within this "application".
- We can think of Excel as being an Object that is made up of lots of other sub-objects (e.g. workbook, worksheet, cells, buttons, etc.)
- We will focus on the pair R and Excel, but COM means that we can connect any number of components together.

DCOM

- The D in DCOM stands for Distributed.
- It allows for R and Excel to be running on different machines and still communicate.
- So we might have R on a high performance compute server and Excel on the user's laptop.
- Distributed means security and permissions are issues. DCOM provides facilities for specifying these.
- DCOM is Windows-specific. It is very similar to CORBA - the Common Object Request Broker Architecture which is platform-neutral.

RDCOMClient package

- From within R, we want to create a new instance of Excel and a worksheet with form elements.
- The RDCOMClient package provides functionality in R
  - to create DCOM objects,
  - get and set their data fields, called PROPERTIES
  - and call their methods, like R functions but specific to a COM class and instances of that class.

RDCOM Basics

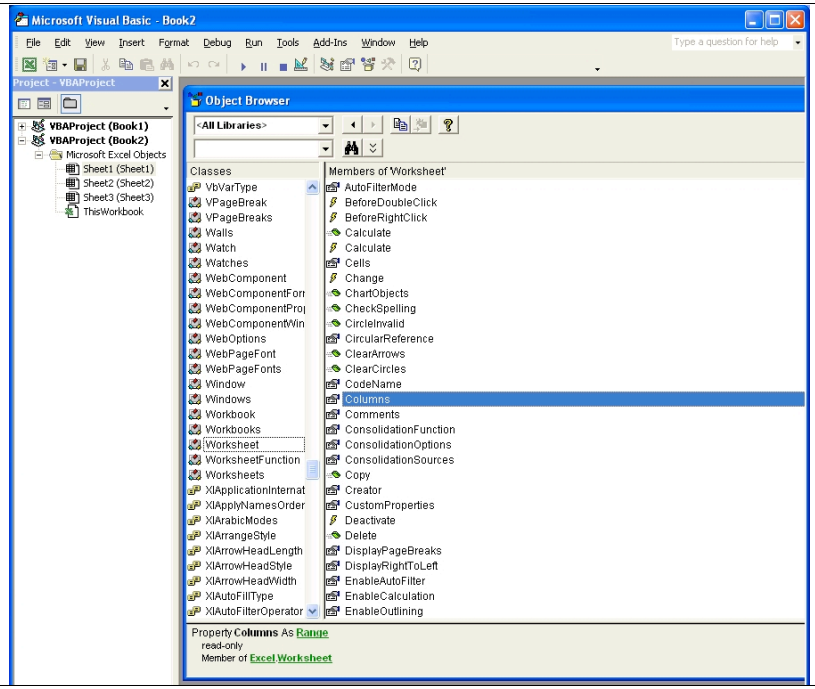
- COMCreate() creates a new instance of a COM object.
- ex = COMCreate("Excel.Application")
  word = COMCreate("Word.Application")
  ie = COMCreate("InternetExplorer.Application")
  or any other registered COM object.
- This gives us a reference or handle to an object that represents that DCOM server.
  It has class "COMIDispatch".
- We can also connect to existing instance - getCOMInstance("Excel.Application")

DCOM Basics

- Each DCOM server provides methods/functions and properties/data that can be accessed by a client.
- These can return other DCOM objects, each with their own methods and functions.
- We can access all of these from within R
  but we have to know which functions and methods are of interest.
- How do we find out what properties and methods are available ?

Discovering properties and methods

- Read books and tutorials on the Web
  Excel 2003 VBA Programmer's Reference
  MSDN website - msdn.microsoft.com/library/
- For Microsoft Applications (and others) use the Object Browser in the Visual Basic Editor.
  Tools -> Macro -> Visual Basic Editor
  followed by View -> Object Browser
  (or Alt + F11 followed by F2 keys)
- Use R commands to query the Type Library of the DCOM object.



SWinTypeLibs

- The SWinTypeLibs package for R allows us to explore information about DCOM classes.
- LoadTypeLib() and getFuncs() are the two main functions for our purposes.

```
lib = LoadTypeLib(ex)
names(lib) # to get the names of the classes in the library
lib[["Application"]]
els = getFuncs(lib[["Worksheet"]])
names(els)
els[['Range']   # PropertyGetDescription with 2 arguments
```


Application Basics

- The different Microsoft Application DCOM objects (Word, Excel) share a similar object model.
- For example, one can make them visible by setting the Visible property to TRUE, and invisible by setting to FALSE.
- In R, we can get the value of a DCOM property in an object via the [[ ]] operator, obj[["propertyName"]], e.g.
  ex[["Visible"]]
- And we can set the value via
  ex[["Visible"]] = TRUE
- Note that values are returned as R objects and R values can be assigned to DCOM properties.

Excel

- The Excel.Application object has a hierarchical structure and the application object is at the top. We navigate through the hierarchy to get at the objects and values we want.
- The application provides its own functions and properties.
- Properties: Visible, Interactive, Height, Memory, ActiveSheet, ...
- Methods: Calculate, Quit, Run, ...
  utility functions such as CheckSpelling, InchesToPoints

General Hierarchy Elements

- Application has a collection of Workbooks
- Within each workbook, there is a collection of Worksheet objects in a Worksheets list.
  These are accessed via the tabs below the sheets.
- Within each Worksheet, we have a rectangular array of cells.
- Collections of cells are described by Range objects.

Workbooks

- The Application has lists of workbook objects that can be accessed via the Workbooks property.
  books = ex[["Workbooks"]]
  or
  books = ex$Workbooks()
- books is an RDCOM object with its own methods...
- It is an ordered collection, like a list in R.
- books$Count() gives the number of elements, books$Item(i) gets the i-th element (starting at 1)

Workbook

- We can add a new workbook via
  book = books$Add()
- The result (in book) is a new Workbook object.
- While you may think this is the thing we want to work with, we are not there yet...
- A Workbook has a list of Worksheet objects.
- sheets = book$Worksheets()
- This has similar methods as Workbooks,
  i.e. Count(), Item()

COMList

- Since the concept of an ordered container of elements arises often in DCOM, the RDCOMClient package provides a type - COMList - to simplify working with these.
- Create a COMList in R using the constructor function
  books = COMList(ex[["Workbooks"]])
- Then, length(books) is an easier version of books$Count()
- books[[i]] gives the i^{th} element of books, equivalent to books$Item(1)

Worksheet & Range

- The Worksheet object is much closer to where we want to be to work with data and forms.
- A Worksheet is made up of cells, and groups of cells are accessed via Range objects.
- Range objects allow us to read and write the values in the spreadsheet, make cells active, editable/uneditable, format values and color regions.
- Can get/set the name of a worksheet via the Name property
  sheet[["Name"]] = "My Data"

Ranges

- Can get the Range for the entire worksheet via
  r = sheet$Cells()
- Can get the rectangular range that covers all the cells actually in use via sheet$UsedRange()
- Can get the rows and columns via sheet[["Rows"]] and sheet[["Columns"]]
- Can get a sub-range using the Range() "method"
  sheet$Range("A10", "C20") - columns A, B, C between rows 10 and 20 inclusive.
- Or sheet$Range(sheet$Cell(10, 1), sheet$Cell(20, 3))

Cell value

- There is no Cell object - everything is done via Ranges.
- To get the value(s) in a range, use the Value property
  range[["Value"]]
- This returns a list in R with as many elements as there are columns in the range.
  Each element is itself a list and contains the values for the cells in that column of the range.
- These are lists in R since each cell may have a different type of data, e.g. string in one, number in another.

- To make use of the values in the cell as numbers, often convenient to use unlist()
- E.g. x = unlist(range[["Value"]])
  or m = matrix(unlist(range[["Value"]]), , 3) for a range with 3 columns.
- Empty cells come across as NULL, so will get dropped in an unlist() call.

Modifying the Range

- We can write Excel spreadsheets by assigning R objects to the Value property of a Range object.

```
r = sheet$Range("A2", "D2")
r[["Value"]] = 1:4
```

- For columns, need use asCOMArray() (!)

```
r = sheet$Range("A2", "A5")
r[["Value"]] = asCOMArray(1:4)
```

- Format cells with different types (dates, currency, etc.) via the NumberFormat property;
Font and color via the Font property;
background colors via the Style property;
Cell borders via the Borders property.

Forms on a Worksheet

- We want to be able to put buttons, pull down menus (combo boxes), spin boxes, check boxes, radio buttons, etc. on the form.
- Two types of forms in Excel - ActiveX objects and internal Excel forms. The latter are slightly simpler, but old and not very general.
- Use ActiveX forms.
- These too are regular DCOM objects, separate from Excel but can be used with Excel.

Adding form elements

- Two approaches to adding form elements
  interactively on the worksheet,
  or programmatically in R.
- Quite easy in R and you will have to do some programming to connect them anyway.
- To work interactively, bring up the "Control Toolbox" toolbar in Excel (Views -> Toolbar -> Control Toolbox)

- Click the button for the desired element type and drag out a region on the worksheet to place it.
- Edit its properties in the Properties window.

Form elements from R

- Can create the elements directly in R.
- Each worksheet keeps a list of these OLE (Object Linking & Embedding) objects via its "OleObjects" property. This is again an ordered collection, so we can use COMList to navigate it.
- The OleObjects value has an Add method to create and add form elements.

```
ole = sheet[["OleObjects"]]
btn = ole$Add(ClassType = "Forms.CommandButton.1",
              Top =20, Left = 30, Width = 70, Height=10)
```

- A button will magically appear on the sheet at those coordinates (in "points")

- The different types of form elements and their names are
  CheckBox, ListBox, ComboBox, CommandButton, Frame, Image, Label, MultiPage, OptionButton, ScrollBar, SpinButton, TabStrip, TextBox, ToggleButton
- The name for the ClassType argument is
  "Forms.<element name above>.1",
  e.g. Forms.ComboBox.1

ActiveX objects

- The object created by ole$Add() is an OLE object, but we talked about ActiveX objects.
- The ActiveX object is inside the OLE object, and we access it via btn[["Object"]] or ole$Item(1)[["Object"]]
- We can now set its properties:
  Caption, Top, Left, Width, Height, LinkedCell, etc.
  Same as on the Properties window in the interactive setting. (And you can use both approaches together.)
- You can now create the interactive form on the worksheet.
  It just won't do anything when you click on the button.

Events

- Some DCOM objects give rise to events,
  e.g. a button and clicking or mouse over
  a worksheet's cell changing, or a worksheet becoming active, or a row being selected.
- These DCOM objects can notify us of their events if we connect an event handler object to them.
- The event handler object must be a DCOM server object which has a particular set of methods that can be called in response to different events
  e.g. Click and MouseDown methods for a button event, Change, Activate, FollowHyperlink for Worksheet.

Creating Event Handlers

- To set an R event handler on a DCOM object, we need to implement the event methods of interest,
  e.g. provide a function for the Click method for button.
- We also need to create a valid DCOM object which uses those R functions when the corresponding method is called by the event source (e.g. the button).
- The RDCOMEvents package in R does this for us, with a few lines of code from us to make it specific to an event source.

RDCOMEvents

- The package dynamically creates a DCOM object from the description of the DCOM class expected by the event source.
- It uses the SWinTypeLibs package to get the description of the expected DCOM object and creates empty functions for the methods we don't supply (and also determines information about the names so that the event source and itself can communicate properly).

RDCOMEvents steps

- Get the type library associated with the event source object
  lib = LoadTypeLib(btn[["Object"]])
- Generate a template server or server description in R
  desc = createCOMEventServerInfo(lib[["CommandButtonEvents"]])
- Add our own handler to the methods
  desc@methods$Click = function(...) { cat("Button's been clicked\n")}
- Create an actual DCOM server object from this description
  server = createCOMEventServer(desc)

Connecting the handler to the events.

- Get the connection source of interest via findConnectionPoint() function
  pt = findConnectionPoint(btn[["Object"]],
  lib[["CommandButtonEvents"]])
- Connect our event handler object to the connection point via Advise()
  Advise(pt, server)
- At this point, you are set to go.
  Generate the events and R will respond.

Code for our Demo- Boostrap computations

```
uniBootstrap =
function(data, statistic = median, B = 999)
{
  n = length(data)
  samples = matrix(sample(data, n * B, replace = TRUE), n, B)
  results = apply(samples, 2, statistic)

  list(samples = samples, results = results)
}
```

Output Results to Excel

```
showBootstrapResults = function(bootData, book) {
  n = nrow(bootData$samples)
  B = ncol(bootData$samples)
    # Create a new worksheet in the book.
  sheet = book$Worksheets()$Add()
    # Put the summary of the results vector in the first row.
  sumy = summary(bootData$results)
  r = sheet$Range("A1:F1")
  r[["Value"]] = names(sumy)
  r = sheet$Range("A2:F2")
  r[["Value"]] = as.numeric(sumy)
    # Put the results vector in the 4th row.
  r = sheet$Range(sheet$Cells(4, 1), sheet$Cells(4, B))
  r[["Value"]] = bootData$results
   #
  for(i in 1:B)  {
    r = sheet$Range(sheet$Cells(6, i), sheet$Cells(6 + n - 1, i))
    r[["Value"]] = asCOMArray(bootData$samples[, i])
  }

  sheet
}
```


Form Event Handler

```
statistics =
list(Mean = mean,
     Median = median,
     Minimum = min,
     Maximum = max)

runBootstrapHandler =
function(...)
{
  B = repetitions[["Object"]][["Value"]]
  stat = statistic[["Object"]][["Value"]]
  data = sheet$UsedRange()[["Value"]]

  results = uniBootstrap(unlist(data[[1]]), statistics[[stat]], B)
  showBootstrapResults(results, book = sheet[["Parent"]])
}
```


Creating the form and Handler

```
library(RDCOMClient)
library(RDCOMEvents)

ex = COMCreate("Excel.Application")
book = ex$Workbooks()$Open("C:/bootstrapTemplate.xls")
ex[["Visible"]] = TRUE
sheet = book$Worksheets()$Item(1)

ole = COMList(sheet[["OleObjects"]])
statistic = ole[[1]] ; repetitions = ole[[2]]
sapply(c("Mean", "Median", "Minimum", "Maximum"),
            function(x) statistic[["Object"]]$AddItem(x))
statistic[["Object"]][["Value"]] = "Mean"

btn = ole[[3]][["Object"]]
lib = LoadTypeLib(btn)
typeInfo = lib[["CommandButtonEvents"]]

connectionPoint = findConnectionPoint(btn, typeInfo)

s = createCOMEventServerInfo(typeInfo, complete = TRUE,
                             methods = list(Click = runBootstrapHandler))
```


Package Information

- R packages available at www.omegahat.org
- Use the Packages menu bar to install packages (or use install.packages).
- Ruuid from BioConductor repository
- RDCOMClient, RDCOMEvents, SWinTypeLibs, RDCOMServer, SWinRegistry from Omegahat repository (www.omegahat.org/R)
- Additional package ExcelUtils (Chris Neff, David James and I) available from Omegahat repository which is less tested but attempts to provide high-level access to some of the ideas we discussed today.

Documentation

- These notes and a paper describing in much more detail what we discussed today are available from www.omegahat.org/RDCOMClient
- Examples at www.omegahat.org/RDCOMClient and for events at www.omegahat.org/RDCOMEvents.
- Discuss Gtk, CGI and DCOM in lectures next week and via Blackboard.
- Please send me comments if you find something wrong in any of the documents, code, etc. or to blackboard if it is not clear.