Introduction to R

Stat 141

R environment, language, ...

- R is three things
  - virtual machine/interpreter
  - a programming language
  - and a collection of built-in and add-on packages
    i.e. collections of useful functions.
- The R virtual machine interprets commands in the R language.
- We will not discuss the interpreter very much, except to look at some of the different ways we can run R commands and control the behaviour of the VM.

R language

- R is a procedural, imperative programming language. executes one command, then another, etc.
- interpreted langauge
  - commands are not compiled into machine code of the chip/CPU
  - instead, a virtual machine (VM) processes each command
  - the virtual machine is a program/executable that interprets the commands and ends up calling compiled code to perform the actions.

R language

- Interactive - don't need to run entire scripts or programs, but can give one command at a time and decide what to do next.
- However, one can combine a collection of commands into a file and have R process the entire file
  R CMD batch filename.R or source('filename.R')
- And can develop and distribute software in R for others to use.
  Vital to making statistics available to non-statisticians and disseminating new methodology.

A little history

- R is a dialect of the S language
  So too is the commercial product S-Plus.
- The S language was developed in Bell Labs over the last 30 years.
  - originally in the 70's-80's by John Chambers, then with Rick Becker and Alan Wilks. (S3 - Blue Book)
  - in early 90's by John Chambers & Trevor Hastie and numerous colleagues and visitors to Bell Labs (Statistical Models - White book)
  - in 93-98, by John Chambers (and DTL) (S4 - Green Book)

ACM Award

- In 1998, John Chambers won the Association of Computing Machinery's Software Award for S
- The citation
  ...forever altered how people analyze, visualize, and manipulate data . . . S is an elegant, widely accepted, and enduring software system, with conceptual integrity
- S is the de facto standard for statistical research and sophisticated data analysis.

Motivation for S

- FORTRAN - Formula Translation language - was developed to allow scientists to express computations to the computer as if they were regular mathematics.
- S grew out of the desire to express statistical calculations and operations to manipulate data and perform data analyses in a natural, convenient, expressive manner that kept the focus on the analysis and thought process of the statistics rather than on "programming" the computer.
- Before S, most analyses involved a lot of FORTRAN code to do basic analysis.

- Standard analysis steps were gathered into reusable libraries of code to avoid having to recreate the code each time or manage the code in different files.
- But in Bell Labs, most analysis were non-standard because they were working on cutting edge problems.
- As a result, there was still a need to write a lot of FORTRAN code and much of it was manipulating data before it could be passed to 3rd party matrix routines.
- S provides an interactive language for managing data, and organizing the order and steps of computations.

- The interpreter
  - reads a command (text)
  - parses it into a data structure that represents the elements and structure of the command internally
  - evaluates the command elements and overall command
  - prints the results (if appropriate)
  - return to wait for next command.
- this is called a REPL - read, eval, print loop.

- S is written in C and much of the built-in functionality is also written in C or FORTRAN.
- And S provides facilities to call code in other languages such as C, FORTRAN, C++, Obective-C, Java, Perl, Python, the Shell, Matlab, ...
- And this allows S to be used as a language to glue low-level computations together in new and different ways specified by each user.
- E.g. can add an extra data transformation step before calling an existing algorithm.

- In addition to being able to call built-in and externally added code in "foreign" (non-S) language, one can also add new functions to R
- and they behave as if they were built-in to the language and behave in exactly the same way as the provided ones.
- So R is very extensible by the user at different levels i.e. within R or via foreign code.

R packages

- Large collection of built-in functions
- Arranged in packages, with each package providing a collection of somewhat related functions for a given type of computation or task.
- R comes with 23 packages and 7 are loaded by default when R starts.
- Other packages provide basic statistical functionality.
- Different repositories or archives provide many other packages
- CRAN, BioConductor, Omegahat, ...
- Over 700 packages publically available.

Graphics

- One of the things that made S successful early on was that it had excellent graphics facilities for statistics.
- It boasted a very flexible and powerful low-level mechanism for creating very rich displays.
- In addition, it provided functions for a large collection of commonly used statistical displays.
- And the plots were aesthetically pleasing, with sensible defaults (e.g. tick marks, labels, titles, colors and glyphs)

- Systems like Matlab caught up with R, even surpassed them in certain types of plots.
- Then trellis was developed by Cleveland, Becker and Shu (Bell Labs) for looking at panels of related plots, based on conditioning and subgroups.
- This is called lattice in R (developed by Deepayan Sarkar) and is available via the library(lattice) package.
- And this is made possible by a very advanced graphics system named `grid' in R developed by primarily Paul Murrell.

Basics of R

- Start R at shell prompt with R (Rgui on Windows or via icon)
- Often run inside a text editor.
- To terminate your R session, call the q() function. answer whether you want to save your session for future use.
- To get help on a function, help("functionName"), or use the help system - help.start() - which displays the pages in your Web browser.

Assignments

- Store results from a command in "variables"
- x= log(10)
- x contains the value of the expression log(10).
- To use the value in a call, just use the name of the variable, e.g.

```
x= 10
log(x)
```

Variable Names

- Variables can be arbitrary names, but
  - can't start with digit or _
  - Shouldn't have quotes, etc. in the name.
- So xyz, x.y.z, .x, abc.lm, abc_lm, x1, x2 are okay.
- 1x, 2x, _x, not okay.
- Avoid names like c, t, s, .C, ...
- # is the character for a comment.
  All text after this on a line is ignored

Assignments

- Older versions of R didn't allow _ in a name.
- Reason: there were three different assignment operators
  - =
  - <-
  - _
- x = 1; y <- 10; z _ 3
- _ is now deprecated.
- Good to know about when reading older scripts.
- Use = or <-

R session

- When we start R for the first time, we have nothing in our work space.
- When we make assignments (at top-level), the variables are stored in our work space.
- objects() tell us what is there. rm() can remove them, etc.
- How do we maintain these across sessions?

rda

- Use save() and save.image()
- save.image() stores the workspace and puts it into a file named .RData
- When R restarts in this directory, it will load the objects from that file if it is present.
- Can avoid this by starting R with
  R --no-restore

save() & load()

- Save.image() saves all objects in the work space.
- May want to save just a few and into a different file name.
- save(x, y, z, file = "myFile.rda")
- Can explicitly load the contents of the .rda file into R using
  load("myFile.rda")
- Note that the .rda files are platform independent
  I.e. you can save on one OS, and use on another.

Basic Calculator

- 1 + 2 - (3*4)/8 - addition, subtraction, ...
- 2^4 - to the power of
- 10%%3 - modulus
- log(10), log(8, 2)

Lanquage

- Essentially, R tries to treat everything as a function call
- log(10), sum(c(1,2,3)), plot(x), ...
- Breaks these up into a function call
  with inputs to the call
  and an output from the call
- Can combine calls with one call as an argument to another.

Exceptions

- What about
  - 1+1
  - 1:10
  - 2^8
- Each of these are actually converted into calls to the associated function.
  - +(1, 1)
  - :(1, 10)
  - ^(2, 8)
- So there really is quite a simple structure

Function Calls

- So the big thing in R is knowing how to call functions
- And R tries to make it convenient.
- The syntax is
  functionName(arg1, arg2, ……)
  or
  functionName(arg1, name=value, name = value)

R Function Calls Features

- R allows for
  - Default values
  - "partial name matching"
  - Variable number of arguments, i.e. … mechanism
  - Lazy evaluation

Default Argument Values

- Function log() has 2 arguments
  log(x, base = exp(1))
- This means that we can call
  log(3)
  and it takes the value of base to be the result of exp(1).
- rnorm has 3 arguments –
  rnorm(n, mean = 0, sd = 1)
- The call rnorm(10, sd = 5) means
  n is 10 and sd is 5 and the unspecified value of mean is taken to be 0.

Default Values

- Not present in C, Fortran, Java.
- Convenient, because we leave out the ones we don't need to specify.
- R matches the arguments by order or by name. If we skip over an argument, we either use the name for the ones we are specifying, or leave a "blank" argument
- rnorm(10, sd = 5) or rnorm(10, , 5)

Named Arguments

- We can always use names for arguments
  - rnorm(10, sd = 5)
  - rnorm(10, , 5)
  - rnorm(10, mean = 0, sd = 5)
  - rnorm(n = 10, mean = 1, sd = 5)
- Which are more readable? Which is easiest to type?
- Depends on whether we are doing interactive, one-time computations or writing software that we want to be able to read later or have others read.

Lazy Evaluation

- One interesting aspect of R is "lazy evaluation".
- This means that the actual expression for evaluating the value of an argument does not happen until the variable is needed.

Objects and the Workspace

- objects(), ls() tell us about what variables we have.
- Where does R find functions like sum(), plot(), objects()
- Like the shell, it uses an "ordered path" in which it looks for *variables, including functions*.
- search()
- Shows a collection of libraries/packages that are available to the session.

Search path

- search()

```
[1] ".GlobalEnv"        "package:methods"   "package:stats"
[4] "package:graphics"  "package:grDevices" "package:utils"
[7] "package:datasets"  "Autoloads"         "package:base"
```

- Work space, S4 object mechanism, statistics functionality, graphics facilities, utilities, datasets, .., built-in functions.
- To find where a variable will be found, use find()
- find("lm")

Manipulating the Search "

- Can load new packages into search path –
  library(lattice)
- Available libraries can be found with command library()
- More available from CRAN, Omegahat, BioConductor, … via
  install.packages()

library() & detach()

- search()

```
[1] ".GlobalEnv"        "package:methods"   "package:stats"
[4] "package:graphics"  "package:grDevices" "package:utils"
[7] "package:datasets"  "Autoloads"         "package:base"
```

- library(rpart)
  search()

```
[1] ".GlobalEnv"      "package:rpart"    "package:methods"
[4] "package:stats"   "package:graphics"  "package:grDevices"
[7] "package:utils"    "package:datasets"  "Autoloads"
[10] "package:base"
```

- detach('package:rpart')
  (or detach(2))
- Ps. Note that we can use " or ` as quotes, but have to use them in pairs.

Everything is an Object

- Functions, data, etc. are all variables in R and we can treat them all (even functions) as inputs to functions.
- Every object has a class
  class(x), class(sum)
  This tells us what type it is.
- Most objects have a length.
  length(c(1, 2)).

Graphics Model

- Devices – different types. Output to
  - a file – postscript(), pdf(), png(), bitmap(), …
  - interactive screen – window(), x11(), gtk(), …
- Can explicitly create a device
  or one will be created when a graphics function needs it.
- Plotting/graphics commands write their contents to the screen or to a file.
- For output to a file, need to explicitly close the device to ensure all content is written –
  dev.off()

Active Graphics Device

- Key point - one and only one device is active at any time.
- All plotting commands work with the currently active device and draw to that device.
- Can have many devices open at any time, but only one is the active device.
- The variable .Devices gives a list of the open devices.
- Can switch between devices using dev.set()
- Can copy contents of one graphics device to another

Example

- Create a histogram with a line for the mean and median.

```
x = rnorm(1000)
hist(x)
abline(v = median(x))
abline(v = mean(x), lwd = 2, lty = 2, col = "red")
```

- Now, copy it to a PDF file.

```
pdf("myHist.pdf") # create new device in a file.
dev.set(dev.prev()) # make previous device active
dev.copy() # copy current to next device
```


Adding to Plots

- When we issue a graphics command, it typically clears the current display on the currently active device.
- Sometimes we want to add to an existing plot.
- Certain graphics functions such as points(), lines(), text(), matlines(), legend(), add content to an existing plot rather than start a new plot.

Graphics Parameters

- Each graphics device has a LARGE set of "current" settings for things such as color, font, text size, margins, line width and style, ...
- We can explicitly set these using the par() function. They take effect in all subsequent graphics commands (for that device).
- We can override individual settings within a single call using the ... mechanism in most graphics functions.

Graphics Parameters


```
par(bg = "gray")
x = rnorm(100)
plot(x)
abline(h = c(-1.96, 1.96), col= "red", lwd= 2)
title("Sample of 100 from N(0, 1)")
idx = which(abs(x) > 1.96)
points(idx, x[idx], pch = "+", col = "red", cex = 2)
```

- This shows setting the background (bg) "globally", col, lwd, pch, cex in individual calls.

Multiple plots

- One can have multiple devices open, and this allows different plots to be compared on the screen in different devices.
- However, we can also have multiple plots on a single device using "screens" or "frames".
- We can partition the device into separate regions (overlapping or not) and draw on each of these in turn.
- Each new plot is drawn on the next frame.

Creating Frames

- par(mfrow = c(nr, nc))
  creates a matrix of screens with nr rows, nc columns.
- layout() can create much more sophisticated and controlled partitions.

Comparisons

- R has functionality for most modern statistical methods since it is written by and for statisticians, along with a large user community of contributors of add-on packages.
- Matlab is focused on engineers, and so lacks much of the functionality and convenience for statistics.
- Weaker programming language, but very fast.
- Also, doesn't have the user community of contributors nor a mechanism for creating & distributing add-on packages.

SAS

- SAS is the most well-know statistics software.
- It deals with large datasets extremely well.
- Focus is on statistics, but increasingly less so and accordingly SAS does not provide all the most modern statistical methods.
- More of a basic language for doing "traditional" statistical analyses.
- The programming language is arcane/akward.
- But a valuable thing to know and widely used along with S.

Perl, Python

- Interpreted, fast scripting languages
  Python is often used interactively also.
- These are programming languages with a much more general audience and purpose than S or Matlab.
- As a result, they are more widely used, but have little built-in functionality for statistics.
- We often use Perl to preprocess data.
- Becoming more widely used in the Bioinformatics worlds BioPerl & BioPython.
- So too is R via BioConductor.

C, C++, Java

- These are for software development rather than interactive scripting.
- Low-level compiled languages that require one to declare variables and their types and create entire programs before running commands.