Part I: Introductory Materials

Introduction to R

Dr. Nagiza F. Samatova
Department of Computer Science
North Carolina State University
and
Computer Science and Mathematics Division
Oak Ridge National Laboratory
What is R and why do we use it?

- Open source, most widely used for statistical analysis and graphics
- Extensible via dynamically loadable add-on packages
- >1,800 packages on CRAN

```r
> v = rnorm(256)
> A = as.matrix(v, 16, 16)
> summary(A)
> library(fields)
> image.plot(A)
> ... 
> dyn.load("foo.so")
> .C("foobar")
> dyn.unload("foo.so")
```
Why R?

- Statistics & Data Mining
- Commercial

Statistical computing and graphics
http://www.r-project.org
- Developed by R. Gentleman & R. Ihaka
- Expanded by community as open source
- Statistically rich

- Data Visualization and analysis platform
- Image processing, vector computing

- Technical computing
- Matrix and vector formulations
The Programmer’s Dilemma

What programming language to use & why?

- Assembly
- Functional languages (C, Fortran)
- Object Oriented (C++, Java)
- Scripting (R, MATLAB, IDL)

Productivity vs. Performance
Features of R

R is an integrated suite of software for data manipulation, calculation, and graphical display

- Effective data handling
- Various operators for calculations on arrays/matrices
- Graphical facilities for data analysis
- Well-developed language including conditionals, loops, recursive functions and I/O capabilities.
Basic usage: arithmetic in R

- You can use R as a calculator
- Typed expressions will be evaluated and printed out
  - Main operations: +, -, *, /, ^
  - Obey order of operations
  - Use parentheses to group expressions
- More complex operations appear as functions
  - sqrt(2)
  - sin(pi/4), cos(pi/4), tan(pi/4), asin(1), acos(1), atan(1)
  - exp(1), log(2), log10(10)
Getting help

• `help(function_name)`
  – `help(prcomp)`
• `?function_name`
  – `?prcomp`
• `help.search(“topic”)`
  – `??topic` or `??“topic”`
• Search CRAN
  – [http://www.r-project.org](http://www.r-project.org)
• From R GUI: Help → Search help...
• CRAN Task Views (for individual packages)
  – [http://cran.cnr.berkeley.edu/web/views/](http://cran.cnr.berkeley.edu/web/views/)
Variables and assignment

- Use variables to store values
- Three ways to assign variables
  - \( a = 6 \)
  - \( a <- 6 \)
  - \( 6 -> a \)
- Update variables by using the current value in an assignment
  - \( x = x + 1 \)
- Naming rules
  - Can include letters, numbers, ., and _
  - Names are case sensitive
  - Must start with . or a letter
R Commands

- Commands can be *expressions* or *assignments*
  - Separate by semicolon or new line
- Can split across multiple lines
  - R will change prompt to + if command not finished
- Useful commands for variables
  - `ls()`: List all stored variables
  - `rm(x)`: Delete one or more variables
  - `class(x)`: Describe what type of data a variable stores
  - `save(x, file=“filename”)`: Store variable(s) to a binary file
  - `load(“filename”)`: Load all variables from a binary file
    - Save/load in current directory or My Documents by default
### Vectors and vector operations

#### To create a vector:

- **c() command to create vector x**
  
  ```r
  x = c(12, 32, 54, 33, 21, 65)
  ```

- **c() to add elements to vector x**
  
  ```r
  x = c(x, 55, 32)
  ```

- **seq() command to create sequence of numbers**
  
  ```r
  years = seq(1990, 2003)
  ```

- **seq() to contain in steps of .5**
  
  ```r
  a = seq(3, 5, .5)
  ```

- **rep() command to create data that follow a regular pattern**
  
  ```r
  b = rep(1, 5)
  c = rep(1:2, 4)
  ```

#### To access vector elements:

- **2nd element of x**
  
  ```r
  x[2]
  ```

- **First five elements of x**
  
  ```r
  x[1:5]
  ```

- **All but the 3rd element of x**
  
  ```r
  x[-3]
  ```

- **Values of x that are < 40**
  
  ```r
  x[x < 40]
  ```

- **Values of y such that x is < 40**
  
  ```r
  y[x < 40]
  ```

#### To perform operations:

- **Mathematical operations on vectors**
  
  ```r
  y = c(3, 2, 4, 3, 7, 6, 1, 1)
  x + y; 2 * y; x * y; x / y; y^2
  ```
Matrices & matrix operations

To create a matrix:

```
# matrix() command to create matrix A with rows and cols
A=matrix(c(54,49,49,41,26,43,49,50,58,71),nrow=5,ncol=2))
B=matrix(1,nrow=4,ncol=4)
```

To access matrix elements:

```
# matrix_name[row_no, col_no]
A[2,1]  # 2nd row, 1st column element
A[3,]   # 3rd row
A[,]    # 2nd column of the matrix
A[2:4,c(3,1)]  # submatrix of 2nd-4th
              # elements of the 3rd and 1st columns
A["KC",] # access row by name, "KC"
```

Element by element ops:

```
2*A+3; A+B; A*B; A/B;
```

Statistical operations:

```
rowSums(A)
colSums(A)
rowMeans(A)
colMeans(A)
apply(A,2,max)
apply(A,1,min)
```

Matrix/vector multiplication:

```
A %*% B;
```
Useful functions for vectors and matrices

- Find # of elements or dimensions
  - $\text{length}(v), \text{length}(A), \text{dim}(A)$
- Transpose
  - $t(v), t(A)$
- Matrix inverse
  - $\text{solve}(A)$
- Sort vector values
  - $\text{sort}(v)$
- Statistics
  - $\text{min}(), \text{max}(), \text{mean}(), \text{median}(), \text{sum}(), \text{sd}(), \text{quantile}()$
- Treat matrices as a single vector (same with $\text{sort}()$)
Graphical display and plotting

- Most common plotting function is `plot()
  - `plot(x,y)` plots $y$ vs $x$
  - `plot(x)` plots $x$ vs `1:length(x)`
- `plot()` has many options for labels, colors, symbol, size, etc.
  - Check help with `?plot`
- Use `points()`, `lines()`, or `text()` to add to an existing plot
- Use `x11()` to start a new output window
- Save plots with `png()`, `jpeg()`, `tiff()`, or `bmp()`
R Packages

- R functions and datasets are organized into packages
  - Packages *base* and *stats* include many of the built-in functions in R
  - CRAN provides thousands of packages contributed by R users
- Package contents are only available when loaded
  - Load a package with `library(pkgname)`
- Packages must be installed before they can be loaded
  - Use `library()` to see installed packages
  - Use `install.packages(pkgname)` and `update.packages(pkgname)`
    to install or update a package
  - Can also run `R CMD INSTALL pkgname.tar.gz` from command line
    if you have downloaded package source
Exploring the *iris* data

- **Load *iris* data into your R session:**
  - `data (iris);`
  - `help (data);`
- **Check that *iris* was indeed loaded:**
  - `ls ();`
- **Check the class that the *iris* object belongs to:**
  - `class (iris);`
- **Read Sections 3.4 and 6.3 in “Introduction to R”**
- **Print the content of *iris* data:**
  - `iris;`
- **Check the dimensions of the *iris* data:**
  - `dim (iris);`
- **Check the names of the columns:**
  - `names (iris);`
Exploring the *iris* data (cont.)

- **Plot Petal.Length vs. Petal.Width:**
  - `plot (iris[, 3], iris[, 4]);`
  - `example(plot)`

- **Exercise:** create a plot **similar** to this figure:

Src: Figure is from *Introduction to Data Mining* by Pang-Ning Tan, Michael Steinbach, and Vipin Kumar
Reading data from files

- Large data sets are better loaded through the file input interface in R.
- Reading a table of data can be done using the `read.table()` command:
  - `a <- read.table("a.txt")`
- The values are read into R as an object of type data frame (a sort of matrix in which different columns can have different types). Various options can specify reading or discarding of headers and other metadata.
- A more primitive but universal file-reading function exists, called `scan()`
  - `b = scan("input.dat");`
  - `scan()` returns a vector of the data read.
Programming in R

- The following slides assume a basic understanding of programming concepts

- For more information, please see chapters 9 and 10 of the R manual:
  http://cran.r-project.org/doc/manuals/R-intro.html

Additional resources

- *Beginning R: An Introduction to Statistical Programming* by Larry Pace
- Introduction to R webpage on APSnet:
  http://www.apsnet.org/edcenter/advanced/topics/ecologyandepidemiologyinr/introductiontor/Pages/default.aspx
- The R Inferno:
Conditional statements

- Perform different commands in different situations
- `if (condition) command_if_true`
  - Can add `else command_if_false` to end
  - Group multiple commands together with braces `{}`
    - `if (cond1) {cmd1; cmd2;} else if (cond2) {cmd3; cmd4;}`
- Conditions use relational operators
  - `==`, `!=`, `<`, `>`, `<=`, `>=`
  - Do not confuse `=` (assignment) with `==` (equality)
    - `=` is a `command`, `==` is a `question`
- Combine conditions with `and` (`&&`) and `or` (`||`)
  - Use `&` and `|` for vectors of length > 1 (element-wise)
Loops

- Most common type of loop is the *for* loop
  - *for* \((x \text{ in } v) \{ \text{loop}_\text{commands}; \}\)
  - \(v\) is a vector, commands repeat for each value in \(v\)
  - Variable \(x\) becomes each value in \(v\), in order
  - **Example:** adding the numbers 1-10
    - \(total = 0; \ for \ (x \text{ in } 1:10) \ total = total + x;\)
- Other type of loop is the *while* loop
  - *while* \((\text{condition}) \{ \text{loop}_\text{commands}; \}\)
  - Condition is identical to *if* statement
  - Commands are repeated until condition is false
    - Might execute commands 0 times if already false
  - *while* loops are useful when you don’t know number of iterations
Scripting in R

- A script is a sequence of R commands that perform some common task
  - E.g., defining a specific function, performing some analysis routine, etc.
- Save R commands in a plain text file
  - Usually have extension of .R
- Run scripts with `source()`:
  - `source("filename.R")`
- To save command output to a file, use `sink()`:
  - `sink("output.Rout")`
  - `sink()` restores output to console
  - Can be used with or outside of a script
Lists

- Objects containing an ordered collection of objects
- Components do not have to be of same type
- Use `list()` to create a list:
  - `a <- list(“hello”,c(4,2,1),“class”);`
- Components can be named:
  - `a <- list(string1=“hello”,num=c(4,2,1),string2=“class”)`
- Use `[[position#]]` or `$name` to access list elements
  - E.g., `a[[2]]` and `a$num` are equivalent
- Running the `length()` command on a list gives the number of higher-level objects
Writing your own functions

- Writing functions in R is defined by an assignment like:
  - `a <- function(arg1, arg2) { function_commands; }`
- Functions are R objects of type “function”
- Functions can be written in C/FORTRAN and called via `.C()` or `.Fortran()`
- Arguments may have default values
  - Example: `my.pow <- function(base, pow = 2) { return base^pow; }`
  - Arguments with default values become optional, should usually appear at end of argument list (though not required)
- Arguments are untyped
  - Allows multipurpose functions that depend on argument type
  - Use `class()`, `is.numeric()`, `is.matrix()`, etc. to determine type
How do I get started with R (Linux)?

- **Step 1:** Download R
  - mkdir for RHOME; cd $RHOME
  - wget http://cran.cnr.berkeley.edu/src/base/R-2/R-2.9.1.tar.gz

- **Step 2:** Install R
  - tar –zxvf R-2.9.1.tar.gz
  - ./configure --prefix=<RHOME> --enable-R-shlib
  - make
  - make install

- **Step 3:** Run R
  - Update env. variables in $HOME/.bash_profile:
    - `export PATH=<RHOME>/bin:$PATH`
    - `export R_HOME=<RHOME>`
  - R
Useful R links

- R Home: http://www.r-project.org/
- R’s CRAN package distribution: http://cran.cnr.berkeley.edu/
- Writing R extensions: http://cran.cnr.berkeley.edu/doc/manuals/R-exts.pdf
- Other R documentation: http://cran.cnr.berkeley.edu/manuals.html