Brief R Tutorial

June 6, 2008

The best way to go through this tutorial is to first install a version of R (see installation section below) and type the commands along with the examples given. This way you can see for yourself what output each command gives.

Contents

1 Introduction

R is a statistical programming language that provides many built in functions for performing statistical analysis. R is also flexible enough to allow users to write their own functions and source code written in a text editor such as NotePad or Emacs. The advantage to learning R is that R is very easy to learn and easy to use. However, R is quite slow when doing heavy computational work (such as Bayesian algorithms) and so using R for heavy computing is not recommended.

This tutorial provides a very *brief* introduction to how to use R. This document will go through some of the most commonly used R functions but will in no way cover all of the functions in R. To learn advanced R functions, you should use Internet searches with the key word *CRAN* which stands for Comprehensive R Archive Network. Using the key word *CRAN* in addition to other key words about the function you are looking for will generally produce a lot of results.

If you forget the specific syntax for the R functions listed in this tutorial, you can simply type ?function and R will return the documentation for function which will provide almost all the necessary information for using function. As an example, typing ?qnorm will return the documentation for the R function qnorm. Alternatively you can use help(functionname) which does the same thing that ?functionname does.

2 R Basics - Installation, Starting, Quitting, and Objects

R can be downloaded and installed for free from the website `http://cran.r-project.org`. This web page provides detailed instructions for installing R on any operating system. If you are using the department computers, you do not need to install R as it has already been installed for you.

Accessing R is different for every operating system. For windows users, simply double click the R icon that is created after installation. For Mac users, you can double click the R icon under your Applications menu. On Linux and the department computers, in the terminal window (if you don't know what a terminal window is then please read the Linux tutorial) simply type R at the command prompt and R will be opened within the terminal window.

When you open R, no matter the operating system you are using, you will see the command prompt symbol ">" which simply means that R is waiting for you to give it a command. To quit R, simply type "q()" in the command prompt and R will ask you if you want to save the workspace before quitting. Generally, it is *NOT* a good idea to save workspace because if you continually save your work space future programs you run may become effected by previously run programs. Nevertheless, it is up to you whether or not you want to save your workspace. After answering this question by typing "y" or "n" R will quit.

R is an object oriented language so output produced from running functions will be output to the screen unless saved as an object. You create an object in R by using the "<-" command. For example, the code

$x <- \text{rnorm}(10,10,10)$

will save the output from using the rnorm() function into x. Having saved the output from rnorm() as the object x, you can print the output from the rnorm() function by simply typing x into the command prompt.

3 Entering or Reading Data Into R

Before you can begin programming you will need to enter or read data into R. This can be done in several ways. You can either type the data in manually (generally *not* a good idea) as a matrix, vector, or array or you can read in your data from a file (much better idea). This section will cover how to do both as you will use both at some point.

3.1 Typing Data In Manually

Here are some commands if you want to create or type in data manually:

- c() - short for "concatenate"; creates an array with a single row of the elements of its arguments. For example, x <- c(1,2,3,4) will create an array object x which contains the numbers 1, 2, 3, 4.
- matrix(data,nrow=,ncol=) - takes the information in data and creates a matrix with nrow rows and ncol columns. For example, the code matrix(c(1,2,3,4),nrow=2,ncol=2) creates a 2×2 matrix with 1 and 2 in the first column and 3 and 4 in the second column. You can create a matrix by rows using matrix(c(1,2,3,4),nrow=2,ncol=2,byrow=T) where the rows will now be (1,2) and (3,4) respectively.
- array(data,dim=c(a,b)) - create from data an array with a rows and b columns. For example array(c(1,2,3,4),dim=c(2,2)) will create a 2×2 array. You can create a higher dimensional array by giving more arguments to the dim argument.
- seq(from,to,by=a) - creates a single row array by creating a sequence from from to to by by. For example, seq(0,1,by=.001) creates a sequence of numbers from 0 to 1 stepping in increments of 0.001.
- diag() - create a diagonal matrix of a single row array. For example, diag(c(1,1,1)) creates the 3×3 identity matrix.
- cbind(a,b) - creates a matrix with the vector a as its first column and b as its second column.
- rbind(a,b) - creates a matrix with the vector a as its first row and b as its second row.
- rep(data,times) - creates a single row array by repeating the numbers in data, times many times. For example, rep(2,3) will create an array by repeating the number 2, 3 times.

3.2 Reading Data From a File

Oftentimes instead of typing in data sets manually, you will want to read data in from a text or other type of file. To do so you can use any of the options below depending on your situation:

- read.table(filename,header=,sep="") - will read the file filename as a data frame (matrix) which is delimited by the character specified in sep="". For example, x <- read.table("mydata.txt",header=T,sep=",") will read the data in mydata.txt which is a comma separated file and has a header (header just means that you have the variable names at the top of the file).
- scan(filename) - this command will do the same as read.table() however scan() will read in the entire file as a single variable. Bottom line, if you have more than one variable in a data set you need to use read.table() because scan() will not distinguish between variables in the file.
- data(dataname) - R has several built in data sets. The data() function will simply load the built in data set. The data will automatically be saved as the object dataname.
- attach() - When you read in data using read.table(,header=T,) or data() and save the data as an object you won't be able to directly access the variables names. For example, say you read in a file mydata.txt which has two variables var1 and var2 using the code x <- read.table("mydata.txt",header=T,sep=" "). Then if you type var1 into the R command prompt it will say that the variable var1 is not found. Instead, the object x has two variable names associated with it, namely var1 and var2. You can access the numbers of var1 by typing x$var1. However, if you type attach(x) and then type var1, the computer will now recognize the variable var1.

3.3 Accessing Elements of Data Arrays, Vectors, or Matrices

Once you have either read data in from a file or created data manually by using any of the above commands, you often need to access a specific element or variable within the object. Here is a brief tutorial about how to access elements of data frames, matrices, and arrays.

- Let x be a single row array. To access the i^{th} element of x type x[i].
- Let x be an $n \times n$ matrix. To access the ij^{th} element type x[i,j].
- Let x be a data frame or any other object with individual variables var1 and var2 contained within it. To access the values saved under the variable name var1 type x$var1. Alternatively, you can attach x and type var1.
- You can see if any object x has any variable names associated with it by typing names(x).

3.4 Determining Sizes of Data Structures/Objects

Often you will need to know the dimension of R objects. For example, you may want to know how many observations are contained in an array. Use any of the following functions to see the dimensions an object.

- length() - returns the length of an object. If the object is an matrix or an array with more than 1 dimension, it will return the total number of elements within that object.
- dim() - returns the dimension of an object. For example, dim(mymatrix) will return the dimension of mymatrix. If mymatrix is a single dimensional array, it will return NULL and you should use length() instead.

4 Basic R Commands You Should Know

Here are list of very basic functions you will use frequently:

- A%*%B - does matrix multiplication of the matrices A and B.
- solve(A) - computes the inverse of the matrix A.
- t(A) - computes the transpose of the matrix A.
- chol(A) - computes the cholesky decomposition of the matrix A.
- eigen(A) - computes the eigenvalues and eigenvectors of the square matrix A.
- apply(A,dim,function) - For a matrix or multi-dimensional array A, apply will perform the function function on the dimth dimension of A. For example, if A is a $p \times k$ matrix, the code apply(A,1,sum) will sum the rows (the first dimension) of A. Alternatively, the code apply(A,2,sum) will sum the columns of A.
- kronecker(A,B) - takes the kronecker product of A and B.

5 Statistical Commands You Should Know

- mean(),median(),max(),min(),sd(),var() - computes the mean, median, max, min, standard deviation, and variance of an object (typically a data array), respectively. If the object is a multidimensional array, the function var() will assume that the columns of the object are different variables and return the variance-covariance matrix.

Distribution	CDF	PDF	Inverse CDF	Draw Randomly
Normal	pnorm()	dnorm()	qnorm()	rnorm()
Binomial	pbinom()	dbinom()	qbinom()	rbinom()
Negative Binomial	pnbinom()	dnbinom()	qnbinom()	rnbinom()
Poisson	ppois()	dpois()	qpois()	rpois()
Beta	pbeta()	dbeta()	qbeta()	rbeta()
χ^2	pchisq()	dchisq()	qchisq()	rchisq()
Exponential	pexp()	dexp()	qexp()	rexp()
F	pf()	df()	qf()	rf()
Gamma	pgamma()	dgamma()	qgamma()	rgamma()
t	pt()	dt()	qt()	rt()
Uniform	punif()	dunif()	qunif()	runif()

Table 1: Table of distribution and their corresponding R functions. Use the help() command to see their exact syntax.

- lm(),glm() - calculate a linear model and generalized linear model for a data set. You will learn A LOT more about these functions in first year courses so no detail is given here. If you want more detail use the help command.
- summary() - summarizes the information of an object. For example if x is a lm object then summary(lm) will summarize the fit of the linear model. Once again, you will learn a great deal more about this function in your first year courses so not a lot of detail is given here.
- **Distributions** - Table 1 displays the basic functions for the most commonly used distributions. Use the help command to see the exact syntax for these commands. For example, help(rnorm) will show you the syntax you need to draw randomly from a normal distribution.

6 Writing Your Own Functions and Sourcing Code

In almost all but the simplest of cases, you will want R to execute a series of commands instead of executing commands line by line as is R's default. To tackle bigger programming jobs, R allows the user to write a series of commands in a separate editor (such as Emacs, TextEdit, or Notepad) and then R will "source" the code and run the code all together. This section will go through the details of how to write your own functions and programs in a separate editor and then source the code in R.

6.1 Writing a Function

You can declare your own function in R using the function() command. The syntax is

```
myfunction() <- function(input1,input2,...){
Function code
}
```

In the above example, the object myfunction() is now a function which takes inputs input1 and input2, and runs the code specified in the Function code. As a specific example, consider

```
mlevar <- function(datavector){
var(datavector)*(n-1)/(n)
}
```

In this case, the function mlevar calculates the MLE of σ^2 instead of the unbiased estimator of σ^2, s^2. Now, the code mle <- mlevar(mydata) will assign to the object mle the value equal to the MLE of σ^2 for your data. If your function is more complex you will need to specify which object calculated in your function to return to the user using the return() function. For example,

```
mlevar <- function(datavector){
output <- var(datavector)*(n-1)/(n)
return(output)
}
```

will return the value in output as the output of the function mlevar.

6.2 For and While Loops

The syntax for writing for loops in R is

```
for(i in 1:N){
Loop Code
}
```

where Loop Code should be substituted with the code you want to run in the loop. The syntax for while loops in R is

```
while(logical argument){
Loop Code
}
```

where *logical argument* should be an argument that is either true or false at each iteration. WARNING R is VERY slow at doing loops. If you are doing a lot of looping in your code, you are advised to use MATLAB or C++ as these languages are MUCH faster than using R.

6.3 Logical Arguments

The following table gives a few logical argument operators used in R.

Code	Interpretation
<	Less than
<=	Less than or equal to
&&	And
>=	Greater than or equal to
>	Greater than

6.4 Sourcing Code and Setting the Working Directory

Once you have completed writing your code in a separate text editor such as Emacs, TextEdit, or Notepad, save the file using the ".R" file extension. For example, "mycode.R". You can then tell R to "source" your code and run the entire program. To do so simply type the command,

```
source("mycode.R")
```

and R will run the entire program. If there are bugs in your code (and there inevitably will be) R will print the error in the R console window for you to return to your code file and make changes.

In the above source example, R will return an error if the file "mycode.R" is not in the current working directory. For example, if R is working from the directory "./Desktop/" and the file "mycode.R" is located in "./home/" directory then you need to set the working directory R is working out of. To do so, use the command setwd("directorypath") where directorypath is the folder you want R to work out of. If you are using unix or a department machine, you can avoid this by simply changing the directory to the desired directory and then opening R (for more on this see the Unix tutorial).

7 Graphics

One of the main reasons that people choose to use R is the great flexibility that R gives in generating graphics. While R can generate a wide variety of graphics, the most commonly used graphics functions are summarized here:

- plot(x,y) - plots the points in x and y on a scatter plot.
- lines(x,y) - plots the points in x and y on the open figure (does not create a new graphic) and connects the points in x and y with a solid black line.
- points(x,y) - adds the points in x and y on the open figure (does not create a new figure as plot() does).
- title("string") - adds the "string" as the title of the figure
- barplot(height,...) - creates a bar plots with heights of the bars equal to the data array height.
- hist(x) - draws a histogram of the data array x
- legend() - adds a legend to the current graphic

You should definitely spend some time messing around with these functions to get practice at generating plots. A good exercise would be to draw a picture of the standard normal distribution.

7.1 Creating a pdf or ps File of an R Figure

R has commands which allow the user to create a pdf or ps file of a figure created in the R environment. To do so use either the pdf() or postscript() commands followed by the dev.off() command. For example, the code:

```
pdf("./myfigure.pdf")
plot(x,y)
dev.off()
```

will create a pdf file of the figure generated by the plot(x,y) command. The dev.off() command tells R that you are done creating the figure and it can now be output the the pdf file.

While you can replace pdf() with postscript(), it is recommended that you create pdf figures because you will most likely be using pdflatex to compile your LaTeXdocument.

8 Writing Output To A File

Having finished running a program, you can then output a data array or matrix to a text or other file type using the write.table() command. As an example, the code write.table(mydata,file="filename.dat") would write the data in the object mydata to the file "filename.dat." This command is especially useful for outputting, say, draws from a posterior distribution drawn using Metropolis-Hastings algorithms.

9 Installing Packages

R has many packages available (free of charge) that will expand the basic function package of R to include more complex functions. For example, R has no default function that draws from a multivariate normal distribution. However the package mvtnorm has a function rmvnorm() that will draw from a multivariate normal distribution

To install a package, type install.packages("packagename"). Once you have typed this, R will prompt you to choose an installation mirror from which to download the packages. Pick a mirror (it doesn't matter which one). R will then automatically install the package for you. Once you install a package you do not need to install it ever again.

Just installing a package is not enough to access the functions available in a package. You also need to source the functions in that package by typing library(packagename) and R will load the functions available in that package for you.

While courses will generally talk to you about what packages to install, the packages that are most commonly used within the department are:

- The coda package - loads functions for calculating convergence diagnostics
- The mvtnorm package - loads functions to draw from a multivariate normal and multivariate t-distribution.
- The xtable package - loads functions to output a table generated in R to latex code.
- The R2WinBugs package - loads functions which will output your R code to a WinBugs file for running a Gibbs sampler. More on this in STA 290.