Introduction to R

EEB C119/C219 (Winter 2012)

Christopher C. Strelioff

Overview

Goals for the lectures

- Immediate goals
  - Learn basics of R
  - Learn to use command line/terminal
  - Program discrete time and ODE models
- Bigger picture
  - Computing and mathematical skills are valuable (jobs)
  - Science – manage lots of data
  - Empowering – don't wait for someone else to write the program, you can do it yourself

Why R?

- R is widely used in all areas of biology
  - Many statistics classes employ R
- R is free
  - You can use R for free in Win, Mac, Linux worlds
  - Active users and developers for R
- R is a scripting language with lots of tools
  - Code (programs) are simple text files, quickly executed (no compiling)
  - Package for ODEs, complex stats models already available
  - Plotting, general programming, etc. all easily done

Using R

Many ways to interact with R

- Command line
  - Good for quick calculations, plotting (and learning)
- Scripts + command line
  - Most general approach
  - Repeatable science with scripts
- Special **gui** (graphical user interface) or **ide** (integrated development environment)
  - rstudio is an example

Command line

How do I get to the command line?

- Windows: **command prompt**
- Mac & Linux: **terminal**
- rstudio
  - Has command line in lower left by default
- Why am I doing this to you???
  - Useful skill for advanced computing
  - Translates to python, perl, c, etc ...

Windows

Command prompt

- Find under Accessories or **type cmd** in search box
- Type **R**

Figures

Mac OS X

terminal

- Find under Application -> Utilities
- Type **R**

Figures

rstudio

terminal is here too . . .

- Command line at lower left
- R is already running

Figures

Command line

For quick calculations, learning . . .

- Calculate $\ln(10)$, (natural log)

```
> log(10)
[1] 2.302585
```

- Calculate $\log_{10}(10)$ (base 10)

```
> log(10,base=10)
[1] 1
```

- Calculate 10^2 and $\log_{10}(10^2)$

```
> 10^2; log(10^2,base=10)
[1] 100
[1] 2
```

Command line

Getting help

- Help inline (press **q** to quit)

  `> help()`

- Help in web browser

  `> help.start()`

- Specific function

  `> help(log)`

  or

  `> ?log`

- Search for help

  `> ??log`

Basics in R

The building blocks of R programs

- Today
  - Variables and assignment
  - Vectors
  - Matrices
  - Plots
  - Workspace
  - Intro to scripts, **source** command
- Thursday
  - **for** loops
  - If else
  - Program flow
  - Functions
  - More scripts

Variables and assignment

Saving calculations

- Simple expression

```
> 1 + 1/12
[1] 1.083333
```

- Save as variable `x`

```
> x <- 1 + 1/12
```

- To see value of `x`, type

```
> x
[1] 1.083333
```

or, assign using

```
> (x <- 1 + 1/12)
[1] 1.083333
```

Variables and assignment

Right side then, left side

- Right side evaluated, then assigned to left

```
> (n <- 10)
[1] 10
> (n <- n + 1)
[1] 11
```

- Confusing? This is **common** in programming
- Another way to do the same thing

```
> n <- 10; n <- n + 1
> n
[1] 11
```

Vectors

Collection of items of same mode (data type)

- A collection of numbers

```
> x <- c(5,67,9,7); x
[1]  5 67  9  7
> y <- c(1,2,x,100); y
[1]   1   2   5  67   9   7 100
```

- Other examples

```
> xletter <- c('b','e','j','h'); xletter
[1] "b" "e" "j" "h"
> xlogical <- c(TRUE,FALSE,TRUE); xlogical
[1]  TRUE FALSE  TRUE
```

Vectors

Accessing specific elements of a vector

- Get one (or more) elements of vector

```
> x <- c(5,67,9,7)
> x[1]
[1] 5
> x[c(1,3)]
[1] 5 9
```

- Get all elements **excluding** specified

```
> x[-1]
[1] 67  9  7
> x[-c(1,3)]
[1] 67  7
```

Vectors

Finding elements with given properties

- Are elements greater than 5?

```
> x <- c(5,67,9,7)
> x > 5
[1] FALSE  TRUE  TRUE  TRUE
```

- Get the **indices** of these elements

```
> z <- which(x>5); z
[1] 2 3 4
```

- Select these elements using **which**

```
> (z <- x[which(x>5)])
[1] 67  9  7
```

Vectors

Changing vectors

- Change element 3 to a 47

```
> x <- c(5,67,9,7)
> x[3] <- 47; x
```

```
[1]  5 67 47  7
```

- Add an extra element

```
> x[5] <- 99; x
```

```
[1]  5 67 47  7 99
```

- Change multiple elements at once

```
> x[c(1,2)] <- c(446,51); x
```

```
[1] 446  51  47   7  99
```

Using the colon operator

- Create a vector with all integers from 2 to 9

```
> x <- 2:9; x
[1] 2 3 4 5 6 7 8 9
```

- Create a vector with all integers from 3 to −1

```
> y <- 3:-1; y
[1]  3  2  1  0 -1
```

- Create a vector with all integers from −1 to −3

```
> z <- -(1:3); z
[1] -1 -2 -3
```

Using seq

- General form: `seq(from,to,by,length)`
- Using by

```
> x <- seq(from=1,to=100,by=10); x
 [1]  1 11 21 31 41 51 61 71 81 91
```

- Using length

```
> y <- seq(from=0,to=100,length=5); y
[1]   0  25  50  75 100
```

- Reverse order

```
> z <- seq(from=3,to=0,by=-1.5); z
[1] 3.0 1.5 0.0
```

Using rep

- Typical use

```
> x <- rep(52,3); x
[1] 52 52 52
```

- Repeat pattern

```
> y <- rep(52:54,2); y
[1] 52 53 54 52 53 54
```

- Repeat each element

```
> z <- rep(52:54,each=2); z
[1] 52 52 53 53 54 54
```

Information about vectors

- What can we find out about a vector?

```
> (x <- seq(from=20,to=1,length=5))
[1] 20.00 15.25 10.50  5.75  1.00
> length(x);
[1] 5
> min(x); max(x)
[1] 1
[1] 20
> mean(x);sum(x)
[1] 10.5
[1] 52.5
> sort(x)
[1]  1.00  5.75 10.50 15.25 20.00
```

Vectorized opertations

- Modify all elements of vector

```
> (x<-1:5)
[1] 1 2 3 4 5
> x^2
[1]  1  4  9 16 25
> log(x)
[1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379
```

- Can also add, multiply vectors (careful of 'wrapping')

```
> (y<-11:15)
[1] 11 12 13 14 15
> x+y; x*y
[1] 12 14 16 18 20
[1] 11 24 39 56 75
```

Vectorized opertations

Wrapping

- Careful with vectors of different size

```
> (x<-1:5); (y<-3:5)
[1] 1 2 3 4 5
[1] 3 4 5
> x+y; x*y
[1] 4 6 8 7 9
[1]  3  8 15 12 20
```

- Vectorized operations are faster
- They are also potentially confusing and might give unexpected results – careful!

Matrices

Multidimensional collection of items of same mode (data type)

- Make the following matrix, M:

$$M = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

```
> (M <- matrix(c(1,2,3,4),nrow=2))
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Matrices

- Make the following matrix, M (note position of elements has changed):

$$M = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
> (M <- matrix(1:4,nrow=2,byrow=TRUE))
     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

Matrices

Accessing elements

- Can access individual enteries, columns or rows

```
> (A <- matrix(1:8,nrow=2,byrow=TRUE))
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
> A[1,2]  # get row 1, column 2
[1] 2
> A[1,]   # get first row
[1] 1 2 3 4
> A[,3]   # get third column
[1] 3 7
```

Matrices

Subsets of matrix enteries

- What elements are greater than 5?

```
> (A <- matrix(1:8,nrow=2,byrow=TRUE))

     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8

> which(A>5) # indicies of elements greater than 5

[1] 4 6 8

> A[A>5]     # values of those elements

[1] 6 7 8
```

Matrices

Vectorized operations

- Take the `log` of all elements

```
> (A <- matrix(11:5,nrow=2,byrow=TRUE))
     [,1] [,2] [,3] [,4]
[1,]   11   10    9    8
[2,]    7    6    5   11
> log(A)
         [,1]     [,2]     [,3]     [,4]
[1,] 2.397895 2.302585 2.197225 2.079442
[2,] 1.945910 1.791759 1.609438 2.397895
```

- Why did I get two `11`'s in matrix `A`? Why is the order backwards?

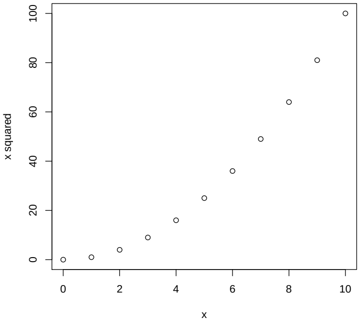
Plots

- Plots are an important part of science and modeling
- There are **many** ways to plot in R
- I provide a series of (basic) examples
  - You should play with plotting
  - Try plotting many ways to best convey information
- Look online for examples of what is possible

Plots
Example 1

- Plot x vs x^2:

```
> x <- seq(from=0,to=10,by=1); y <- x^2
> plot(x,y,xlab="x",ylab="x squared")
```

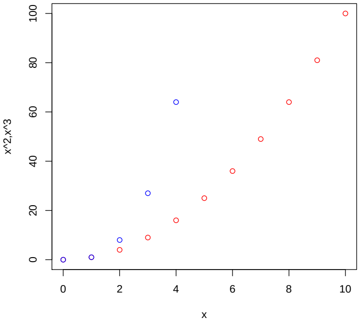


Plots

Example 2

- Plot with points (default), x^2 and x^3

```
> x <- seq(from=0,to=10,by=1); y <- x^2; z <- x^3
> plot(x,y,xlab="x",ylab="x^2,x^3",col="red")
> points(x,z,col="blue")
```

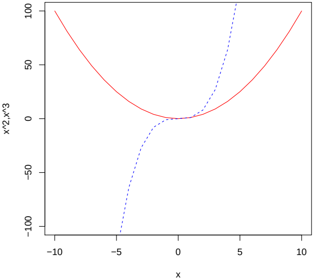


Plots

Example 3

- Plot with lines and modify plotting range

```
> x <- seq(from=-10,to=10,by=1); y <- x^2; z <- x^3
> plot(0, type="n", xlab="x",ylab="x^2,x^3",
+      xlim=c(-10,10),ylim=c(-100,100))
> lines(x,y,lty=1,col="red")
> lines(x,z,lty=2,col="blue")
```



Workspace

R has a memory

- R remembers assignments that have been made in a session
- R can also remember **between** sessions
  - Is this good or bad?
  - Depends on what you want to do . . .
- Example:

```
> x <- 5;  y <- c(4,98)
> ls()
[1] "x" "y"
```

Workspace

How do we clear/delete?

- Often it is a good idea to clean R's memory
- Delete single item

```
> x <- 5;  y <- c(4,98)
> rm(x); ls()
```

```
[1] "y"
```

- Clear entire workspace

```
> x <- 5;  y <- c(4,98)
> rm(list=ls()); ls()
```

```
character(0)
```

Scripts

Motivation

- An R script is a text file with a series of commands
- Why use scripts?
  - Makes R coding reusable (by you and others)
  - Easy to make **lots** of comments
  - Remember how code works (you will forget)
  - Easier to manage big projects
  - Use hoffman2
- We will be using scripts all the time!

Scripts

Example - Geometric Model

- Our first discrete time model was the geometric model:

$$n_{t+1} = Rn_t$$

- We can solve this by hand

$$n_t = R^t n_0$$

- Let's generate data for n_t and plot it!

Scripts

How to create and run

- Creation
  - Create in plain text editor (notepad, TextEdit, etc)
  - Do **not** use Word or similar programs – hidden formatting!
  - Can use editor in **rstudio**
- Running
  - Use **source** command: `source(scriptname)`
  - Command line or terminal
  - rstudio

Geometric Model

```
# 2012, Jan 24th  ---- Chris Strelioff
#   * Plot the geometric model
#     n(t+1) = R * n(t)   , for t=0 to t=10
#   * Parameters:
#     R = 1.2   "Reproductive number"
#   * Initial condition:
#     n(t=0) = 10
## clear workspace
cat('\n','* Clearing Workspace','\n'); rm(list=ls())

## set my working directory and save location -- change 'path-to-your-directory'
setwd('path-to-your-directory'); mydir <- getwd()
cat('\n','* Working directory set to: ', mydir, '\n')

# make vector of times
cat('\n','* Generate vetor of times', '\n')
show(time <- 0:10)

# set parameters and IC
R <- 1.2;n0 <- 10
cat('\n', '* Setting parameters and IC', '\n  R=', R, '\n  n[0]=', n0, '\n')

# use vectorized operation to evaluate n(t) at these times
#  - employ solution n(t) = R^(t) * n(0)
cat('\n','* Generate values for n(t)', '\n'); show(nn <- (R^time)*(n0))

# generate plot and save as pdf
cat('\n','* Plotting', '\n')
pdf('GeometricModel.pdf')  # set ouput filename -- this startouput to file
plot(time,nn, xlab="time t", ylab="N(t)", col="red")
dev.off() # this stops output to file
```