Computational Methods in Astrophysics

Dr Rob Thacker (AT319E)

thacker@ap.smu.ca

Yet more on R

- More useful things for scripting:
  - Functions
  - Matrices and eigenvectors
  - User input
  - Intermediate graphics

Functions

- R is in essence a functional language
- So describing functions is a key part of programming
- The format is simple, but hides potential complexity

```
myfunc <- function (arguments) {
        code using arguments
        perhaps other things
        return(var)
}
```

- You don't have to have the return statement

Functions II

- A simple example – mysquare

```
mysquare <- function(n) {
n*n
}
```

- Try `mysquare(3)`
- You can also have a temporary variable

```
mysquare <- function(n) {
a = n*n
return(a)
}
```

Functions - Exercises

- Let's have some fun! Define a function that calculates the series (10 mins)

$$\sum_{i=1}^{n} \frac{x^i}{i!}$$

- Need two inputs, x and n(>0)
- HINT: don't use a loop!
  - Define vectors to do things for you (`sum()` helps ;))
  - Factorial function is just `factorial()`

Functions - Exercises

- One possible solution

```
myfunc <- function(x,n) {
1+sum(x^(1:n)/factorial(1:n))
   }
```

- Can check with myfunc(1,8) should be close to e (2.718…)

Functions - Exercises

- Let's do something involving if () in the function
- Define a function myfunc2 (10 mins)

$$f(x) = \begin{cases} x^2 + x + 3 \; if \; x < 0 \\ 2x + 3 \; if \; 0 \leq x < 2 \\ x^2 + 2x - 1 \; if \; x \geq 2 \end{cases}$$

- It should return a vector of values
- Use this to plot the function on -4 to 4

Functions - Exercises

- You might be tempted to try:

```
myfunc2 <- function(x) {
len = length(x)
for (i in 1:len) {
y=x[i]
if (y<0) {
    y = y^2 + y + 3
} else if (0<=y & y<2) {
        y = 2*y + 3
} else {
        y = y^2 + 2*y - 1
}
x[i]=y
}
return(x)
}
```

- Will work – but more compact form is possible

Vector ifelse

- `ifelse(test,x,y)` returns elements from `x[i]` if `test[i]` is `TRUE`, and `y[i]` if `test[i]` is `FALSE`

  Example, try:

```
a=c(1,2,3,4)
ifelse(a %% 2 == 0,"even","odd")
```

- Can use this to produce an f(x) function that is more compact

Vector ifelse

- Since we have three parts of the function, we can nest `ifelse` statements

```
myfunc3 <- function(x) {
ifelse(x < 0, x^2 + x + 3, ifelse(x < 2, 2*x+3, x^2+
2*x - 1))
}
```

- This also shows how functions can be treated as vectors too
- A very neat and compact solution
- For functions with two parts only the second part need appear in the ifelse

Matrices

- To set up a matrix, use the matrix command:

```
m = matrix(1:9, ncol=3)
```

- Check the format, and then try

```
m = matrix(1:9, ncol=3, byrow=TRUE)
```

- That will fill up by rows rather than cols
- To transpose, use the `t()` function

```
trans.m = t(m)
```

- Element-wise operations +,-,*/

Matrices

- True matrix multiplication is defined by
- `m %*% trans.m` (for example)
- Identity matrix short-hand: `I <- diag(3)`
- Determinant: `det(m)`
- Inverse: `inverse.m <- solve(m)`
- Check `inverse.m %*% m = identity` (to machine prec)
- `solve(m,b)` would solve mx=b

Eigenvalues & eigenvectors

- Define a matrix `m = matrix(1:9 %% 4, ncol=3, byrow=TRUE)`

  `y <- eigen(m)`

  `y$val` are the eigenvalues

  `y$vec` are the eigenvectors
- Check these work – i.e. check eigenvector equation
- You'll need to get the vector indexing right ☺

User input

- `scan()` is a simple way of entering vector data
- `x1 <- scan()` will create a vector of numerics
  - Double return to finish
- Try entering strings – what happens?
- `x2 <- scan(what=" ")` allows you to enter strings
- You can also use it to read from files – will return a vector rather than data frame like read.table

Readline

- Alternative to using scan – here's a wonderful example of it's use:

```
fun <- function() {
ANSWER <- readline("Are you a satisfied R user? ")
if (substr(ANSWER, 1, 1) == "n")
  cat("This is impossible. YOU LIED!\n")
else
  cat("I knew it.\n")
}
if(interactive()) fun()
```

- The `substr` function just extracts from `ANSWER` between `start(=1)` and `stop(=1)`

Text processing

- Yes – you can grep!
- Try `my.str <- "Hi there"`

  `grep("th",my.str,value=TRUE)`
- If you don't include value then grep returns the index of the matching vector
- There are a number of other functions too, check out

`http://www.regular-expressions.info/rlanguage.html`

Data frames

- Start the console, create a script and enter

```
height <- c(1.7,1.65,1.34,1.5,1.8)
name <- c("Izzy","Chris","Mel","Viv","Alex")
mass <- c(70,55,50,62,80)
eyes <- c("brown","green","brown","blue","brown")
hair <- c("brown","blonde","blonde","blonde","brown")
ourpop <-data.frame(name,eyes,hair,height,mass)
```

More on plots

- The plot command takes one data-set input – but usually need more than 1 data set
- So define height2 = c(1.54,1.72,1.55,1.6,1.8)
- Replot the ourpop$height,ourpop$mass dat
- You can add height2 (using same mass values) as follows:

```
points(height2,ourpop$mass,col="2")
legend("topleft",c("Heights","New Height"),
col=c("1","2"),pch=c(21,21))
```

Error Bars

- Use the `arrows()` command
- Define upper and lower values in y, and the corresponding x value
- Call with (example)

```
arrows(xupper,yupper,xlower,ylower,col=1,angle=90,length=0.1)
```

Summary

- Learn how to use function definitions, they can really make things easier
- The inbuilt linear algebra is really quite effective and simple to use
- User input can be done in a number of ways, scan and readline offer different alternatives