R Introduction 2

J.M. Ponciano

August 17, 2007

1 Input/Output

1. Output to screen:

   You can either type the object name, print it using `print` or the function `cat`

2. Output to file:

   - x is an object, ASCII:
   - x is a matrix or data.frame, ASCII:

3. Input from file (ASCII files):

   - rectangular data file:
   - irregular shaped files
   - loading long script or function:

2 Matrix operations and functions

1. basic operations

   - dimensions
   - simple functions

- matrix arithmetic (be careful with recycling!!)
- transposing
- crossproducts:
- diagonal extraction or assignment
- sweeping operations:

```
> A <- matrix(1:12,nrow=4,ncol=3)
> colmean <- apply(A,2,FUN=mean)
> Centered.A <- sweep(A,2, colmean, FUN="-")
> A
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
> colmean
[1]  2.5  6.5 10.5
> Centered.A
     [,1] [,2] [,3]
[1,] -1.5 -1.5 -1.5
[2,] -0.5 -0.5 -0.5
[3,]  0.5  0.5  0.5
[4,]  1.5  1.5  1.5
```

- Outer product of verctors `a` and `b`: all possible products of elements of the data vector of `a` with those of `b` :

```
> a<-A[,1]
> b <-A[,2]
> a
[1] 1 2 3 4
> b
[1] 5 6 7 8
> outer(a,b)
     [,1] [,2] [,3] [,4]
[1,]    5    6    7    8
[2,]   10   12   14   16
[3,]   15   18   21   24
[4,]   20   24   28   32
```

Now suppose you want to evaluate the function

$$f(x, y) = \frac{\cos(y)}{1 + x^2}$$

over a regular grid of values with $x-$ and $y-$ coordinates defined by the R vectors x and y respectively. You could use:

```
z <- outer(x,y, function(x,y){cos(y)/(1+x^2)})
```

2. Linear/matrix algebra

   - Inversion:

   - Solving system of equations:

   - Cholesky decomposition:

   - Eigenanalysis:

3 List and factor functions

1. To produce absolute one-way or higher dimensional frequency tables for a factor vector or vectors:

```
> x<-factor(c("Red","Red","Green","Red"))
> y<-factor(c("Short","Tall","Short","Short"))
> table(x)
x
Green   Red
    1     3
> table(x,y)
       y
x       Short Tall
  Green     1    0
  Red       2    1
```

2. List component operations

3. Creating a list from a numeric vector using a factor vector:

```
> y<- c(18,17,19,15,16)
> sex <-factor(c("m","f","m","f","f"))
> split(y,sex)
$f
[1] 17 15 16

$m
[1] 18 19

> boxplot(split(y,sex))
```

4 Character functions

Generally operate on entire character strings, not on single characters, useful for titles and axis labeling.

- Concatenation
- Counting characters
- Extracting part of a character string:

5 Control structures

1. `if` and "else" blocks

```
# try these with different values for y
x = 1;y=10;
if(y<5){x=2}
x
```

2. `ifelse` function:

3. Multiple branches with `switch` function:

6 Looping

1. `for` loops

```
#For loop syntax: for (var in seq) {expression}

 for (i in 1:3) {print(i)} #basic for loop

 for (i in c(1,3,17)) {print(i)} #sequence need not be in order
```

```
for (char in c("A", "Z", "2")) {print char} #sequence need not
be numbers

forseq <- 1:4
for (i in forseq) {print i} #sequence can be constructed
elsewhere
```

2. `while` loops

```
i<-0
while (i<4) # basic while loop
            # logical check done before any operations
{
   i<- i+1
   print(i)
}
```

3. `repeat` loops

```
i<-0
repeat      #basic repeat loop
            #logical check after operations
{
   i=i+1
   print(i)
   if (i>3) {break}
}
```

4. Think twice before looping, try to_____.

```
# Ex. 1: row means:
x <- matrix(rbeta(10000,3,2),2000,5)
mean.vec <- rep(0,2000) # create space in advance
```

```
#--looping
now <- proc.time()[1:2];
for(i in 1:2000) mean.vec[i] <- mean(x[i,])
speed<- proc.time()[1:2] - now;
speed

# -- vectorizing:
now <- proc.time()[1:2]
mean.vec <- apply(x,1,mean)
speed <- proc.time()[1:2]-now
speed

rm(mean.vec,x,speed)
```

7 Function writing basics

1. argument assignment
2. explicit vs. implicit returns from function
3. debugging/file editing
   -
   -
   -
   -
4. communicating/ error checking with the function user
   -
   -
   -
   -
5. More sophisticaded debugging
   -
   -
6. Calling C and Fortran routines

8 Examples

```
# Random numbers generators examples:

# Uniform:
a<- runif(n=10000,min=0,max=1);
hist(a,xlab="Histogram of 10000 pseudo-random numbers from a U(0,1) distribution")

# Binomial

b <- rbinom(n=10000, size=10,prob=0.75);
hist(b,xlab="Histogram of 10000 draws from X~Binom(n=10,p=0.75)",col=3)

# Bernoulli

bernoulli.draw1 <- rbinom(n=1000, size=1,prob=0.75);
bernoulli.draw2 <- sample(c(0,1), size=1000, replace=T, prob = c(0.25,0.75));

par(mfrow=c(2,1));
hist(bernoulli.draw1,  xlab="");
hist(bernoulli.draw2,  xlab="");

# A very important example:

# The long tedious way:
par(mfrow=c(2,4));
hist(rbinom(n=10000,size=5,prob=0.45),main="n=5, p=0.45",xlab="",col=3)
hist(rbinom(n=10000,size=7,prob=0.45),main="n=7, p=0.45",xlab="",col=3)
hist(rbinom(n=10000,size=9,prob=0.45),main="n=9, p=0.45",xlab="",col=3)
hist(rbinom(n=10000,size=11,prob=0.45),main="n=11, p=0.45",xlab="",col=3)
hist(rbinom(n=10000,size=20,prob=0.45),main="n=20, p=0.45",xlab="",col=3)
hist(rbinom(n=10000,size=50,prob=0.45),main="n=50, p=0.45",xlab="",col=3)
hist(rbinom(n=10000,size=100,prob=0.45),main="n=100, p=0.45",xlab="",col=3)
hist(rbinom(n=10000,size=10000,prob=0.45),main="n=10000, p=0.45",xlab="",col=3)

# The short version:
ntrials <-c(5,7,9,11,20,50,100,10000);
len     <- length(ntrials);
par(mfrow=c(2,4));
for(i in 1:len){

hist(rbinom(n=10000,size=ntrials[i],prob=0.45),xlab="", main=paste("p=0.45, n=", sep="",toString(ntrials[i])),col=4);
}

# Compare to (CLT!):

binom.hist <- rbinom(n=10000,size=10000,prob=0.45);
```

```
normal.hist<- rnorm(n=10000,mean=10000*0.45, sd= sqrt(10000*0.45*0.55));

par(mfrow=c(2,1));
hist(binom.hist, xlab="", xlim=c(4300,4700),col=2, main="Binomial sample, n=10000, p =0.45");
hist(normal.hist,xlab="",xlim=c(4300,4700),col=2, main="Normal sample, mean = 10000*0.45");
```

A little function example: For many organisms, births occur in regular, well-defined breeding seasons. Discrete time population growth models are adequate for these organisms. Let N_t denote population size of one of these organisms at time t. Here's a little function that outputs a time series of exponential population growth according to the model:

$$N_{t+1} = N_t e^a.$$

```
exp.growth <- function(a,len,no){
# This function returns a time series of length 'len' of population growth
# according to the model N(t+1) = N(t) * exp(a);
# the argument 'a' is the growth rate and 'no' is the initial population size
# The specified length has to be >= 2

pop.vec     <- rep(0,len);
pop.vec[1] <- no;
for(i in 1:(len-1)){
pop.vec[(i+1)] <- pop.vec[i]*exp(a);
}
return(pop.vec)
}

# trying the function:
popsim <-exp.growth(a=0.05,len=20,no=2)
popsim

> popsim
 [1] 2.000000 2.102542 2.210342 2.323668 2.442806 2.568051 2.699718 2.838135
 [9] 2.983649 3.136624 3.297443 3.466506 3.644238 3.831082 4.027505 4.234000
[17] 4.451082 4.679294 4.919206 5.171419
```

When writting functions, documenting them so that anyone can use them is very important. In fact, repeatability or how easy it is to reproduce one's work is a key ingredient of doing good science. Here are some advices from Mark on how to document a function.

```
#DOCUMENTATION:
#     Documentation is the most important feature of a function!
```

```
#     Any text in a line to the right of a # sign is a comment and will not
#     influence function execution.
#     The MINIMUM DOCUMENTATION
#     THAT IS SOCIALLY RESPOSIBLE
#      5 key elements 1) Description of arguments
#     pased into the function. 2) Description of values or ojects returned
#     by the function.  3) Coders name.  4) Date last modified.
#     And, 5) explicit description of side effects (if any) the function has.
#     Side effects are any changes to the environment outside of the function
#     other than through the function's return value.  Documentation complexity
#     should increase with function complexity and the likelihood that others
#     will need to understand the function.
#     Other useful documentation elements include: a statement of function
#     purpose, a variable dictionary, a modification history, a list of
#     dependencies, an algorithm description, and an algorithm source.
#     Comments can be inserted in the body of the function to clarify
#     tricky or critical steps.
```