The R Book

Chapter 2: Essentials of the R Language
Session 3 (2_2)

Generating regular sequences (Q last week)

Integers

```
> 1:5
[1] 1 2 3 4 5

> x<-(1:5)
> x
[1] 1 2 3 4 5

> y<-c(1,2,3,4,5)
> y
[1] 1 2 3 4 5

>2*1:5
[1]  2  4  6  8 10

s5 <- rep(x, times=5)
[1] ????
```

Generating regular sequences (Q last week)

Integers

```
> 1:5
[1] 1 2 3 4 5

> x<-(1:5)
> x
[1] 1 2 3 4 5

> y<-c(1,2,3,4,5)
> y
[1] 1 2 3 4 5

>2*1:5
[1]  2  4  6  8 10

s5 <- rep(x, times=5)
[1] ????
```

For floating points, use seq

default grammar : seq(from,to, by)

```
> seq(1.1,1.5, by=.1)
[1] 1.1 1.2 1.3 1.4 1.5
```

or specify :
- length
- from
- to
- by

```
> seq(length=5, from=1.1, by=.1)
[1] 1.1 1.2 1.3 1.4 1.5
```

Summary Information from Vectors by Groups (repetition, last week)

In the terminal / Console :

data<-read.table("/home/michael/Documents/.../daphnia.txt",header=T)
data → ...
names(data) → "Growth.rate" "Water" "Detergent" "Daphnia"
tapply(Growth.rate,Detergent,mean) → error

attach(data) (you must « attach » the data to R in order to work with it)
tapply(Growth.rate,Detergent,mean) →

```
  BrandA   BrandB   BrandC   BrandD
3.884832 4.010044 3.954512 3.558231
```

the mean growth rate for each detergent

> tapply(Growth.rate,list(Detergent,Water),mean) →

```
            Tyne     Wear
BrandA 3.661807 4.107857
BrandB 3.911116 4.108972
BrandC 3.814321 4.094704
BrandD 3.356203 3.760259
```

the mean growth rate for each detergent and water type

Summary Information from Vectors by Groups (repetition, last week)

In the terminal / Console :

data<-read.table("/home/michael/Documents/.../daphnia.txt",header=T)
data → ...
names(data) → "Growth.rate" "Water" "Detergent" "Daphnia"
tapply(Growth.rate,Detergent,mean) → error

attach(data) (you must « attach » the data to R in order to work with it)

tapply(Growth.rate,Detergent,mean) →

```
   BrandA    BrandB    BrandC    BrandD
3.884832  4.010044  3.954512  3.558231
```

the mean growth rate for each detergent

> tapply(Growth.rate,list(Water,Daphnia),median) → `?????`

the median growth rate for water type and Daphnia clone

In RStudio :
Import Daphnia.txt (/home/michael/Documents/.../daphnia.txt
attach(Daphnia)
...

Using « with » rather than « attach »

Problem with « attach »
problems with resolving names (end up with multiple copies of the same variable name)

Alternative, 'functionwise' specification of the data
- lm or glm functions have a '*data=*' argument

… or use « with »

detach(data)

tapply(Growth.rate, list(Detergent,Water),mean) → Error
with(data,tapply(Growth.rate, list(Detergent,Water),mean)) → ???

Example datasets in « datasets »
search()
library(help=datasets), have a look for « OrchardSprays »
or just : data()

with(OrchardSprays,boxplot(decrease~treatment)) → ????

with(mammals,plot(body,brain,log="xy")) → ????

Using « with » rather than « attach »

```
reg.data<-read.table("/home/michael/.../regression.txt",header=T)
```

```
  growth tannin
1     12      0
2     10      1
3      8      2
4     11      3
5      6      4
6      7      5
7      2      6
8      3      7
9      3      8
```

Using « with » rather than « attach »

```
with(reg.data,{
+    model<-lm(growth~tannin)
+    summary(model)})
```

Groups of statements to which the « with » function applies are contained within curly brackets.

lm, linear model function : looked in reg.data for the variables "growth" and "tannin"

with : function used reg.data for constructing the environment

reg.data : contains the data of regression.txt

Alternative :

```
summary(lm(growth~tannin,data=reg.data))
```

Using « with » rather than « attach »

Other examples

calculate the number of ‘no’ (not infected) cases in the bacteria dataframe

```
library(MASS)
with(bacteria,bacteria)

with(bacteria,tapply((y=="n"),trt,sum))
```

plot brain weight against body weight for mammals on log-log axes

```
with(mammals,plot(body,brain,log="xy"))
```

Using attach in This Book

We use attach throughout this book because it makes the code easier

- refer to variables by name, **x** rather than **dataframe$x**
- shorter models, **lm(y~x)** rather than **lm(y~x,data=dataframe)**
- straight to the action, **plot(y~x)** not **with(dataframe,plot(y~x))**

Parallel Minima and Maxima: pmin and pmax

pmin, finds the minimum from a set of variables
pmax, finds the maximum from a set of variables

Example :

x → 0.99822644 0.98204599 0.20206455 0.65995552 0.93456667 0.18836278
y → 0.51827913 0.30125005 0.41676059 0.53641449 0.07878714 0.49959328
z → 0.26591817 0.13271847 0.44062782 0.65120395 0.03183403 0.36938092

pmin(x,y,z) →

0.26591817 0.13271847 0.20206455 0.53641449 0.03183403 0.18836278

pmax(x,y,z) →

0.99822644 0.98204599 0.44062782 0.65995552 0.93456667 0.49959328

Subscripts and Indices

Parts of vectors or elements are selected with « subscripts »

Subscripts have square brackets [2], functions have round brackets (2)

Subscripts on vectors, matrices, arrays and dataframes :
One set of square brackets [6], [3,4] or [2,3,2,1]

Subscripts on lists :
double squarebrackets [[2]] or [[i,j]]

Two subscripts to an object (matrix, dataframe)
The first subscript refers to the row number
the second subscript refers to the column number

Convention in R : blank = all
- [,4] means all rows in column 4 of an object
- [2,] means all columns in row 2 of an object

$: another indexing convention in R,
to extract named components from objects

Data Types

R has a wide variety of data types :

- scalars
- vectors (numerical, character, logical)
- matrices, arrays (multidimensional)
- data frames
- lists

Vectors

```
a <- c(1,2,5.3,6,-2,4) # numeric vector
b <- c("one","two","three") # character vector
c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE) #logical vector
```

Data Types

R has a wide variety of data types :

- scalars
- vectors (numerical, character, logical)
- matrices, arrays (multidimensional)
- data frames
- lists

Matrices (two dimensions)

Columns must have the same mode (numeric, character, etc.) and length

```
mymatrix <- matrix(vector, nrow=r, ncol=c, byrow=FALSE,
   dimnames=list(char_vector_rownames, char_vector_colnames))
```

byrow=TRUE : matrix is filled up row by row, ... =FALSE : filled by columns

Data Types

R has a wide variety of data types :

- scalars
- vectors (numerical, character, logical)
- matrices, arrays (multidimensional)
- data frames
- lists

Data Frames

related to matrix, but columns can have different modes

```
d <- c(1,2,3,4)
e <- c("red", "white", "red", NA)
f <- c(TRUE,TRUE,TRUE,FALSE)
mydata <- data.frame(d,e,f)
names(mydata) <- c("ID","Color","Passed") # variable names
```

Data Types

R has a wide variety of data types :

- scalars
- vectors (numerical, character, logical)
- matrices, arrays (multidimensional)
- data frames
- lists

Lists

An ordered collection of objects (components), allows to gather a variety of (possibly unrelated) objects under one name.

A list with 4 components, a string, a numeric vector, a matrix, and a scalar

```
w <- list(name="Fred", mynumbers=a, mymatrix=y, age=5.3)
```

Working with Vectors and Logical Subscripts

Example : a vector containing the 11 numbers 0 to 10:
>x<-0:10
> x<-0:10

add up the values
> sum(x)
[1] 55

count cases, how many are below 5 ?
> sum(x<5)
[1] 5

This works because of « coercion » :
Logical TRUE has been coerced to numeric 1
Logical FALSE has been coerced to numeric 0

```
 >x<5
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
        1     1     1     1     1     0     0     0     0     0     0
```

Working with Vectors and Logical Subscripts

count cases, how many are below 5 ?

```
> sum(x<5)
[1] 5
```

This works because of « coercion » :

Logical TRUE has been coerced to numeric 1

Logical FALSE has been coerced to numeric 0

```
 >x<5
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
        1     1     1     1     1     0     0     0     0     0     0
```

To find the sum of the values of x that are less than 5:

```
> sum(x[x<5])
[1] 10
```

Alternatively :

```
> sum(x*(x<5)) → 0x1 + 1x1 + 2x1 + 3x1+ 4x1 + 5x0 + 6x0 + 7x0 +
                 8x0 + 9x0 + 10x0
[1] 10
```

work out the sum of the three largest values in a vector

1) sort the vector into descending order
2) add up the values of the first three elements

```
y<-c(8,3,5,7,6,6,8,9,2,3,9,4,10,4,11)

> sort(y)
 [1]  2  3  3  4  4  5  6  6  7  8  8  9  9 10 11

> rev(sort(y))
 [1] 11 10  9  9  8  8  7  6  6  5  4  4  3  3  2

> rev(sort(y))[1:3]
[1] 11 10  9

> sum(rev(sort(y))[1:3])
[1] 30
```

Addresses within Vectors, which()

```
> y
 [1]  8  3  5  7  6  6  8  9  2  3  9  4 10  4 11

> names(y)<-1:15
> y
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
 8  3  5  7  6  6  8  9  2  3  9  4 10  4 11

> y<-as.vector(y)
> which(y>5)
[1]  1  4  5  6  7  8 11 13 15
```

answer is a set of subscripts

To see the values of y > 5,

```
> y[y>5]
[1]  8  7  6  6  8  9  9 10 11
```

Addresses within Vectors, length(), seq()

```
> length(y)
[1] 15
> length(y[y>5])
[1] 9
```

Extract every nth element from a vector with seq()

> xv<-rnorm(1000,100,10) random numbers, length 1000
mean 100
standard deviation 10

from, to (try 1000) by

> xv[seq(25,length(xv),25)]

```
 [1]  78.55738 102.43888  98.35262  90.35767 108.45821 105.87016 103.74105
 [8]  88.86585 107.17449 107.34596  90.55796 106.00437 119.99036 101.28521
[15]  92.50904  92.89058 105.71306  84.45659 105.17407 100.62878 110.44917
[22]  85.21301 103.60246  85.07047  98.14852  99.96667  78.37900 115.25886
[29] 109.51843  98.88434  83.30707 105.62302  97.21630  90.99227 106.60842
[36]  74.92409  92.88968  90.89482  96.53100  91.82529
```

every 25th value