Introduction to R

- What is R ?
  A language and environment for statistical computing and graphics
- S-Plus is a commercial implementation of the “S” system
- R is based on the “S” system developed by Bell Laboratories
- R is available as Free Software
- Matrix-based programming language with rich statistical features.

- ***Acquiring and Installing R***

  “Windows” : free version of R can be downloaded from http://www.r-project.org;

Outline

- Data Structures
- Functionality
- Input/Output
- Workspace Management

R Basics

- **Note**: everything in R is case sensitive
- Assignments can also made using “=” .
- Variable names may be delimited by a ‘.’

```
> a.meaningful.name <- 6
```

- Indices always begin with 1.
- Comments: #

```
> x<-1+5
> x
[1] 6
> y=c(1,2,3,4)
> y
[1] 1 2 3 4
> z=1:4
> z
[1] 1 2 3 4
> z[1]
[1] 1
>
```

Mathematical Operators

- R as a calculator:

```
> 2 + 3
[1] 5
> 3*4/6 + 2*(1 + 9)
[1] 22
>a <- matrix(1:30, 5,6)
>b<- c(1,2,3,4,5,6)
>a%*%b # matrix multiplication
> t(a) # matrix transpose
```

Built-In R Functions

- R comes with a suite a built-in mathematical and statistical functions.

```
> sqrt(54)
[1] 7.348469
> mean(1:5)
[1] 3
> lm(y~x) # simple linear regression
> ?sum # look for help about function sum
```

Matrices

- **Matrices are 2 dimensional vectors.**

```
> A <- matrix(1:9, nrow=3, ncol=3, byrow=T)
> A
     [,1] [,2] [,3]
[1,]   1    2    3
[2,]   4    5    6
[3,]   7    8    9
> row.names(A) <- c( ‘a’ , ‘b’ , ‘c’ )
>A
  f g h
a 1 2 3
b 4 5 6
c 7 8 9
```

Extracting and Extending Matrices

Extract information from the matrix using indices.

```
> A[,1]
 a b c
 1 4 7
```

```
> A[1,]
f g h
1 2 3
```

Extend the matrix by adding rows or columns.

```
> B <- cbind(A, c(-10,-20,-30))
>B
  f g h
a 1 2 3 -10
b 4 5 6 -20
c 7 8 9 -30
```

```
>C <- rbind(A, c(-10,-20,-30))
>C
  f   g   h
a 1   2   3
b 4   5   6
c 7   8   9
 -10 -20 -30
```

A matrix can only consist of the one data type; e.g. numeric, character

Interrogating a Matrix Object

- Useful functions are:

```
> dim(A)
[1] 3 3
> ncol(A)
[1] 3
> nrow(A)
[1] 3
> length(A)
[1] 9
```

Similarly for a vector object:

```
> length(x)
```

Operating on Matrices

- A really useful function for matrices is the apply function. This allows us to apply a specific function to row-wise or column-wise.

```
> apply(A, 1, mean)
[1] 2 5 8 # the 1 means row-wise,
# use 2 for column-wise
```

Data Frame

- A data frame is a collection of column vectors.

	Gpdh	Sod	Xdh	AvRate	Myr
Drosophila	1.50	25.7	30.4	22.4	55
Fungi	40.0	24.9	13.7	21.4	300
Human	13.2	19.2	19.2	17.5	600

- A useful way to store table-like information.

```
> molclock <- data.frame(Gpdh=c(1.50, 40, 13.2),
+ Sod=c(25.7, 24.9, 19.2), Xdh=c(30.4, 13.7, 19.2),
+ AvRate=c(22.4, 21.4, 17.5), Myr=c(55, 300, 600),
+ row.names=c( “Drosophila” , “Fungi” , “Human” ))
```

Extracting data from data frame

Extracting data from a data frame object by column, we can use indices or names:

```
> molclock[,1]
[1] 1.5 40.0 13.2
> molclock[,” Gpdh” ] or molclock$Gpdh
[1] 1.5 40.0 13.2
```

For rows: we must use row indices.

```
> molclock[2,]
         Gpdh  Sod   Xdh   AvRate  Myr
Fungi    40    24.9  13.7    21.4   300
> class(molclock[,1])
[1] “numeric”
```

Recall: a *data.frame* object is a collection of column vectors.

```
> class(molclock[2,])
[1] “data.frame”
```

List Structures

- Up until now, all our data structure objects have needed a uniform data type. List structures are powerful because we can store multiple data types in the same object.

```
> My.Objs <- list( “item1"=c(1.3, 99.6, 2.45),
+  “item2"=matrix(rnorm(100), nrow=10),
    “item3"=molclock)
```

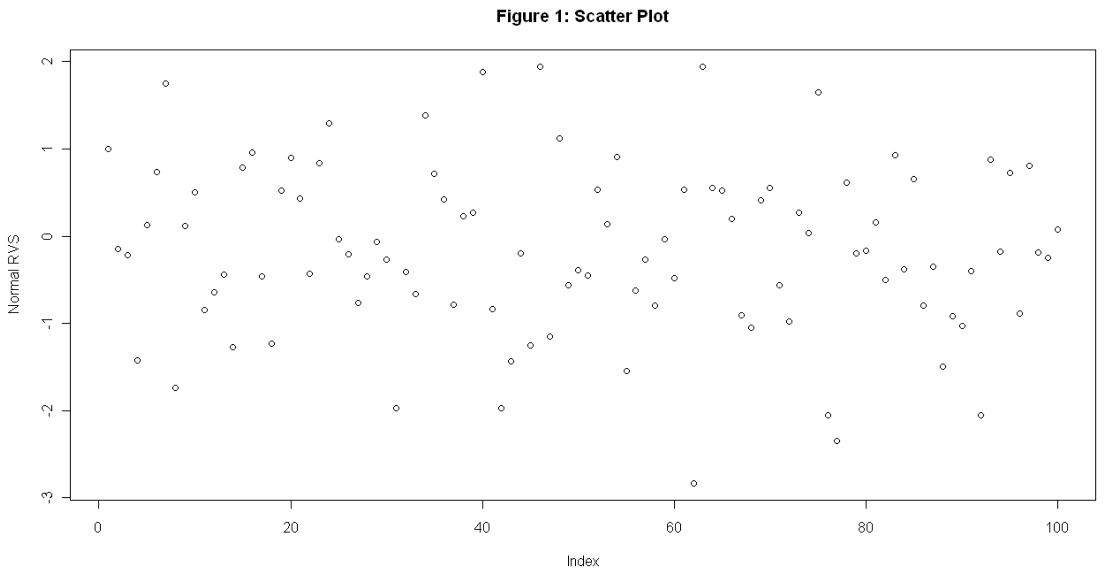
We extract data from a list using names or indices.

```
> names(My.Objs)
[1]  “item1"  “item2"  “item3"
> My.Objs$item1
[1] 1.30 99.60 2.45
> My.Objs[[1]]
[1] 1.30 99.60 2.45
```

Visualizing Data: Plot Function

- A simple scatter plot:

```
> x.dat <- rnorm(100) # 100 N(0,1) rvs
> plot(x.dat, xlab="Index", ylab="Normal RVS",
+ main="Figure 1: Scatter Plot")
```



Exporting Graphics

In Windows:

• right mouse click to copy to clipboard.

For most operating systems:

```
> bitmap("file.bmp")
> plot(cars$speed)# <- insert code for making plot here
> dev.off()
```

You can create export graphics to many file formats – bitmap, jpeg, gif, postscript, etc.

Classes

- A class describes the way an object in R is stored.
  Strings: “Homo sapiens”
  Numeric: 3.141593
  Boolean: TRUE, FALSE
- We can interrogate an object to find out its class:

```
> a <- FALSE
> class(a)
[1] "logical"
> is.numeric(a)
[1] FALSE
```

- Classes also reflect their data structure, eg. matrix, data.frame, function.

```
>class(molclock[,1])
>class(molclock[2,])
```

Missing Values

- NA is the all-inclusive symbol for a missing value in R.

```
> mean(c(1, 4, NA))
[1] NA
> mean(c(1, 4, NA), na.rm=T)
[1] 2.5
```

- We can test whether an object is a missing value.

```
> NA == NA
[1] NA # this doesn't work!
> is.na(NA)
[1] TRUE
> na.omit(c(1, 4, NA))
[1] 1 4
```

- Other objects: NaN, Inf. Inf and -Inf are positive and negative infinity

whereas NaN means ‘Not a Number’

For Loops

- For loops are very simple in R.

```
> for( m in 1:3 ){
+ print(m)
}
[1] 1
…
> for( m in c( “DNA", “RNA", “Protein") ){
+ print(m)
}
[1] “DNA"
…
```

- Note: R does not process for loops very quickly, try to avoid them for large data if you can (eg. Use apply)

Conditional Statements

- We can use conditional statements to automate tasks and functions.
- If..Else Block

  If(condition 1 holds) then do task 1. Else, do task 2.

```
> if( x > 0 ){ print("positive") }
+ else{ print("negative") }
```

- While Block

  While(condition 1 holds) then do task 1. If condition 1 no longer holds,stop.

```
> while( x > 0 ){ x <- x + rnorm(1) }
```

- You can put the break command inside an if(…) to break out of the conditional loop.

Writing Your Own Functions

- Imagine you need to write a simple function that returns both the mean and the standard deviation of a vector in a list structure.

```
> mean.and.sd <- function(x){
+ res.mean <- mean(x)
+ res.sd <- sd(x)
+ res = list(mean=res.mean, sd=res.sd)
+ return(res)
+ }
> mean.and.sd(rpois(10,5))
```

- You can use the args function to find out what arguments a function needs.

```
> args(mean.and.sd)
[1] function (x)
NULL
```

Inputting Data into R

- **If the input file format looks like:**

V1 V2 V3 …

** 1   2   4 ….**

** 2   1   2 ….**

….

```
>my.data<-read.table(“data.txt”,header=TRUE,
   sep=“\t”)
>summary(my.data)
>dim(my.data)
>names(my.data)
```

- **Other read-in functions**: read.csv, scan, readLines

Outputting Data from R

To output data to a simple table text file, we can use write.table.

```
>write.table(cars,file="cars.txt",sep="\t",row.names=FALSE)
```

The cars.txt looks like

```
"speed" "dist"
     4      2
     4     10
     7      4
     7     22
     8     16
     9     10
    10     18
    10     26
```

Other write functions: write, cat.

Workspace Management

- Where am I?

```
> getwd() # returns the working directory
> setwd("C://") # sets the working directory
> dir() # lists files in working directory
> list.files()
```

- How can I tell what objects I have?

```
> ls()
```

- To remove individual objects use rm():

```
> rm("name.of.object")
```

- To save specific objects use save():

```
> save(x, file="fileName.Rdata")
```

- At a later date, you can load this into your workspace:

```
> load("fileName.RData")
```

Libraries

- Libraries are a collection of R functions that together perform a specialized analysis or task.
- For example: genetics package.

CRAN Description:

Classes and methods for handling genetic data. Includes classes to represent genotypes and haplotypes at single markers up to multiple markers on multiple chromosomes. ….

- Consult CRAN for more: http://cran.us.r-project.org/

Install package

- Install packages from CRAN:

  Use the menu: “Packages” – “Install package(s)”

  Select a CRAN mirror, e.g. USA (CA1) and click OK

  Install packages: e1071, neural, randomForest, SOM, VR

  Install packages from bioconductor:

  Visit http://www.bioconductor.org/

  Click “Install-How To”

  If want to install a subset of most frequently used packages, use the installation script:

source("http://www.bioconductor.org/biocLite.R")

biocLite()

Load library

- Load the library

¬ *library(MASS)*

- *To check the content of a library:*

>library(help=MASS)

Helpful Functions

- To boot up HTML help files:

```
> help.start()
```

- To pop up a help file on an individual function.

¬ help(function)

¬ ?function

- To search for help on something around a topic or function:

```
> help.search("plot")
```

- To search on a string for something:

```
> apropos("string")
```

- Reference:
- 统计建模与R软件，薛毅、陈立萍，清华大学出版社