Oxford Genomics CENTRE

UNIVERSITY OF OXFORD

Introduction to R Programming

26th February 2018

Helen Lockstone and Ben Wright
Bioinformatics Core

wellcome centre human genetics

Further examples and exercises

The following slides are from a previous version of this course and were created by Rafik Salama and Helen Lockstone

R as a calculator

Arithmetic operators

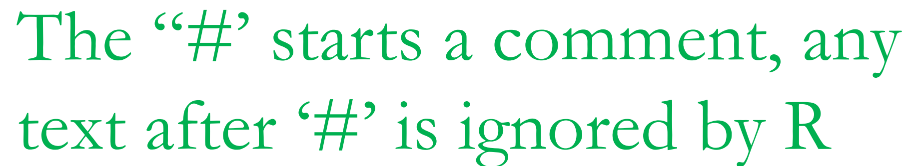
+	add
–	subtract
*	multiply
/	divide
^	raise to the power

Slide courtesy of Rafik Salama

R as a calculator

How to enter numbers

```
> 10^3 # this computes 10 * 10 * 10
[1] 1000
> 1e3 # this just enters "1000" in scientific notation
[1] 1000
> 10e2
[1] 1000
> -5e3
[1] -5000
> -5e-2
[1] -0.05
> -5e-10
[1] -5e-10
```



Slide courtesy of Rafik Salama

R as a calculator

Special numbers that are NOT numbers

Missing values are NA, Not Available

```
> x <- c(0,2,4,NA)
> x
[1]  0  2  4 NA
> mean(x)
[1] NA
> mean(x, na.rm=T) # na remove true
[1] 2
```

Slide courtesy of Rafik Salama

R as a calculator

Something totally different is NaN, Not any Number

Results from errors

```
> 0/0 # division by zero
[1] NaN
```

Like infinity, Inf

```
> 1/0
[1] Inf
> exp(1e200)
[1] Inf
```

Slide courtesy of Rafik Salama

R as a calculator

Some powers

```
> (1/27)^(1/3)
[1] 0.3333333
```

Let's do something useful: body mass index

```
> bmi <- 75/1.75^2
> bmi
[1] 24.48980
```

$$bmi = \frac{weight}{height^2}, \text{units}\left[\frac{kg}{m^2}\right]$$

We can also round the output to a given number of decimal places:

```
> bmi <- round(bmi, 1)
> bmi
[1] 24.5
```

Slide courtesy of Rafik Salama

R as a calculator

- All the usual functions exist (log, exp, sin, cos, tan, sqrt, etc.)

```
> log(10)
[1] 2.302585 #so log() is the natural logarithm
> exp(1)
[1] 2.718282 # this is e
> pi
[1] 3.141593 # π is known as pi
> sin(pi)
[1] 1.224606e-16 # should be zero and is fairly
close
> cos(pi)
[1] -1
```

Slide courtesy of Rafik Salama

Exercise: R as a calculator

- Calculate and assign result into variable with name z
  - Square root of 2
  - Sine of 2π
  - Natural logarithm of 10 and 10^{1000}
  - 2^{10}
  - e^{10}
  - Calculate the values of the following functions

$$\frac{1}{9},\quad \left(\frac{1}{9}\right)^2,\quad \sqrt[3]{27},\quad \left(\frac{1}{9}\right)^{\frac{1}{2}},\quad \sqrt{\frac{1}{9}}$$

Slide courtesy of Rafik Salama

Common data structures (object types)

- Vectors
- Factors
- Matrices
- Dataframes
- Lists

- Objects can be created in many different ways
- We'll work through some examples of each type and look at ways to access or manipulate the data contained within an object
- Beware that the type (class) of an object and data it contains (numeric, character etc) may affect how it is treated by R

Slide courtesy of Rafik Salama

R basics: constructing vectors

Many ways to make vectors

Simplest is the colon ":"

```
> x <- 5:1
[1] 5 4 3 2 1
> x <- 0:4
[1] 0 1 2 3 4
```

More versatile is the concatenate function "c"

```
> x <- c(0,2,-1,pi,10)
[1]  0.000000  2.000000 -1.000000  3.141593 10.000000
> x <- c(0:5,10:15)
 [1]  0  1  2  3  4  5 10 11 12 13 14 15
```

Slide courtesy of Rafik Salama

R basics: constructing vectors

Sequences with finer control than “:” (colon operator) with seq

```
> x <- seq(0,2,.25)
[1] 0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00
> length(x)
[1] 9
> x <- seq(0,2,length=11)
[1] 0.0 0.2 0.4 0.6 0.8 1.0 1.2 1.4 1.6 1.8 2.0
```

Repeating elements with rep

```
> rep(1:2,4)
[1] 1 2 1 2 1 2 1 2
```

Slide courtesy of Rafik Salama

R basics: working with vectors

- `> x <- 1:10`
- `> x`
- ` [1]  1  2  3  4  5  6  7  8  9 10`
- `> max(x); min(x)`
- `[1] 10`
- `[1] 1`
- `> range(x)`
- `[1]  1 10`
- `> mean(x)`
- `[1] 5.5`
- Similarly, we can find sum(x), sd(x), var(x) etc

R basics: Vector operations

- Functions will be applied element-wise to the vector they are operating on:

```
> x2 <- x^2
> x2
 [1]   1   4   9  16  25  36  49  64  81 100
> sqrt(x2) # back to the original vector
 [1]  1  2  3  4  5  6  7  8  9 10
> y <- c(1, 2, 3, 4, 5)
> x * y
[1]  1  4  9 16 25  6 14 24 36 50
```

- Can you see what R has done here? (print x and y objects to the screen again if needed)

R basics: Sequences of Numbers

- Using a function such as `seq()` to generate vectors of numbers is very flexible, and introduces the notion of arguments (information that is passed to the function)
- `> help(seq) # shows the help page for this function`
- For `seq()` there are five possible arguments that can be used in different combinations; often a default value for each argument is defined in the code for the function (and described in the help page) along with examples of using that particular function
- from, to, by, length, along
- If not specified by name, they are assumed to occur in this order

R basics: Sequence generation

- The following 4 expressions are all equivalent:

```
> 1:30
> seq(1, 30)
> seq(from=1, to=30)
> seq(to=30, from=1)

> seq(-5, 5) # R interprets -5 and 5 as the first 2
arguments of seq() (i.e. from, to) and uses the default
by=1 in the lack of specific assignment for this argument
```

- A different interval can be assigned either implicitly as the 3rd argument or explicitly by name (in which case can be placed in any order)

```
> seq(-5, 5, 0.2)
> seq(-5, 5, by=0.2)
> seq(5, -5, by=0.2) # order of sequence reverses
```

R basics: Shortcut functions

- A related function is `rep()`, which can be used for replicating an object in various ways:

```
> x <- 1:3
> rep(x, times=5)
[1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
> rep(x, each=5)
[1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
> y <- 8:10
> rep(c(x, y), times=2)
[1]  1  2  3  8  9 10  1  2  3  8  9 10
>c(rep(x, each=2), rep(y, each=2))
[1]  1  1  2  2  3  3  8  8  9  9 10 10
```

- Syntax, especially bracket placement, can get complicated - inspect the resulting object to check it created what you wanted!

R basics: vector subscripts (indices)

- Accessing blocks of vectors with subscripts
  - Distinguish index from value of a vector element
  - You access some elements of vectors rather than all of them using square brackets [ ] next to the variable name:

```
v <- -50:50 # make simple example vector
mean(v) # this uses all elements of v
v[1] # first element
v[40:60] # a section in the middle
v[length(v)] # last element since length() gives number of elements
v[(length(v)-10):length(v)] # use the last 10 elements (maybe you want to select the final stage)
```

Slide courtesy of Rafik Salama

R basics: vector subscripts (indices)

- Something special: you can select elements of a vector that fulfil a logical condition

```
v[v > 0] # all elements for which condition is true will be selected
v[v == 50] # exactly equal, may give no hit
v[v == mean(v)] # also unlikely to get a hit
```

R basics: Plotting

- Plot a function to see it's shape

```
x <- seq(-2*pi,2*pi,length=101) # make vector x
y <- sin(x) # calculate vector y as a function of x
plot(x,y)
plot(x,sin(x)) # another way of doing the same
```

- The result is OK, but for publication quality this default plot is not good enough, we want lines not points (type="l"), y-axis labels horizontal (las=1), etc.
- Inspect the plots side by side to see the effect:

```
> par(mfrow=c(1,2))
> plot(x,y)
> plot(x,y, las=1, xlab="x", ylab="sin(x)", type="l",
main="Trigonometric functions")
> par(mfrow=c(1,1))
```

- Many more parameters can be set within the plot() call

Slide courtesy of Rafik Salama

R basics: Plotting

- Distinguish high-level from low-level plotting functions
  - High-level plotting functions create a new plot, complete with axes, ticks, labels etc.
    - These are difficult to change afterwards!
    - Make plot behave by setting graphics parameters before the call to plot, e.g.
      - `par(las=1)`
    - Or set them within the high-level plotting function, e.g.
      - `plot(x,y, las=1, ...)`
    - Example high-level plotting functions
      - plot(), qqplot(), hist(), image(), contour(), wireframe(), cloud(), etc.

Slide courtesy of Rafik Salama

R basics: Plotting

- Low-level plotting functions add to an existing plot
- Let's plot sin(x) again
  - `plot(x,y, las=1, xlab="x", ylab="f(x)", type="l")`
- Now that we have a plot, we can add
  - `lines(x,cos(x),col="red")`
  - `points(x, cos(x)+rnorm(length(x), mean=0, sd=0.1), col="blue") # cos(x) with some noise added`
  - `title("Trigonometric functions")`
  - `text(0.5*pi+pi/4, sin(0.5*pi), "sin(x)")`
  - Example low-level plotting functions: lines, points, text, title, legend, grid, arrows, etc.

Slide courtesy of Rafik Salama

R basics: Plotting

- Here is a list of options for `plot()` that are commonly used (there are far more). Many can also be used in other plotting functions

```
> xlim=c(0,80), ylim=c(0,1) # set x and y axis limits
> log="x", log="y", log="xy" # logarithmic axes
> asp=1 # aspect ratio of axes (y/x), use 1 for square axes
> type="whatever" # l for lines, p for points, b for both, n for
  nothing, o for both overplotted, s for stairs
> col="yellow" # never use yellow, it is very hard to see
> lwd=1 # line width
> lty=1 # line type: 1 solid, 2 dashed, etc up to 6
> pch=1 # plot character, can use integers in range 0:25 or
(easier) a single character in quotes, e.g. pch="+", pch="*",
pch="? "
```

- See `help(plot)` for more details

Slide courtesy of Rafik Salama

Exercise: vectors

- Make these vectors
  - 0.00 0.25 0.50 0.75 1.00
  - 0 7
  - 1 2 3 1 2 3 1 2 3
  - Make a vector x containing numbers 0,1,2,...,10
  - Calculate vectors y according to these equations

$$y = x^2, \quad y = \sqrt{x}, \quad y = \frac{1}{x}, \quad y = \frac{1}{x^2}$$

  - Make a vector x containing 100 elements from -π to π (equally spaced)
  - Calculate vectors y according to these equations

$$y = 2\sin(x), \quad y = \cos(2x)$$

Slide courtesy of Rafik Salama

Exercise: plotting

- Plot the following functions into a single plot (hint, use the `lines()` function to superimpose additional plots after creating the first with `plot()`
  - Make vector for x axis, from 0 to 2 (make 100 points for smooth curve)
  - Plot $y=x^2$ versus x as a red line
  - Plot $y = \sqrt{x}$ versus x as a blue line
  - Plot y=x versus x as a black line

- We can also add a legend to the plot

```
> legend("topleft", legend=c("y=x^2", "y=sqrt(x)",
"y=x"), col=c("red", "blue", "black"), lty=1)
```

Slide courtesy of Rafik Salama

Exercise: vector subscripts (indices)

- Make a vector from 0 to 99
  - Select these elements
    - first
    - last
    - 10 in the middle
    - the last 50
  - Select elements based on logical conditions
    - all elements smaller than 10
    - all elements smaller and equal to 10
    - all elements equal to 10
    - all elements equal to 100
    - all elements equal to 0.1

Slide courtesy of Rafik Salama

R basics: Character vectors

- Character vectors are often used in R, for example as plot labels – they are simply strings (words) in quotes
- They can be concatenated into a vector with the `c()` function
- `paste()` is another very useful function for creating character strings with flexible formatting

```
> samples <- 1:10
> labs <- paste(c("X", "Y"), samples, sep="")
> labs # note the c("X", "Y") vector (of length=2) is recycled 5 times to match the length of the sequence 1:10
```

R basics: Character vectors

- Look at how the following commands affect the character vector that is generated

```
> labs <- paste("Patient", samples, sep="")
> labs <- paste0("Patient", samples)
> labs <- c(paste0("Patient", samples),
"Patient11")
```

- `gsub()` as a way to modify the content of the string

```
> labs <- gsub("Patient", "Sample", labs)
> labs <- gsub("Sample", "Sample_", labs)
```

R basics: Factors

- Factors are useful for handling categorical data

```
> groups <- rep(c("WT", "MU"), each=3)
> groups
> class(groups) # character vector
> groups <- as.factor(groups) # coerce to factor
> class(groups) # factor
> groups
[1] WT WT WT MU MU MU
Levels: MU WT
> table(groups) # useful summary function giving the
number of entries for each level of the factor
> groups <- factor(groups, levels=c("WT", "MU")) # re-
order the levels (default alphabetical)
```

R basics: Matrices

- Matrices (or more generally arrays) are multi-dimensional generalizations of vectors – vectors that can be indexed by two or more indices. For example, a vector of length 20 could be made into a 2-dimensional matrix of e.g. 10 rows by 2 columns or 5 rows by 4 columns.

```
> x <- 1:20
> x <- matrix(x, ncol=2, nrow=10)
> dim(x) # get the dimensions
[1] 10  2
> head(x) # shows the first 6 rows
> colnames(x) <- c("Col_1", "Col_2")
> x
```

Slide courtesy of Rafik Salama

R basics: matrix subscripts (indices)

- Accessing blocks of matrices with subscripts using [ ]
  - Arrays with 2 dimensions (2D array or 2D matrix)
    - 1st dimension: rows
    - 2nd dimension: columns
      - That's how we read newspapers
  - Accessing blocks of elements in an 2D array
    - A[1,2]
    - Read like this: [rows , columns]
    - A[,1]
    - Empty means everything (all rows, or all columns)

Slide courtesy of Rafik Salama

R basics: matrix subscripts (indices)

- Construct our example array from a vector

```
> v <- 0:15
> v
 [1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
> A <- matrix(v,4,4)
> A
     [,1] [,2] [,3] [,4]
[1,]    0    4    8   12
[2,]    1    5    9   13
[3,]    2    6   10   14
[4,]    3    7   11   15
```

Slide courtesy of Rafik Salama

Exercise: matrix subscripts

- Now play with this matrix, selecting
  - the entire first row
  - the entire first column
  - the block in the middle
  - the last two entire columns
  - all values larger than 10

Slide courtesy of Rafik Salama

R basics: More data objects

- Lists are a general form of vector in which the various elements need not be of the same type, and are often themselves vectors or lists (see section 6.1 of R manual)

- Data frames are like matrices but the columns can be of different types – they can contain columns of numerical data and others with categorical variables, so are often useful for describing experiments. We'll look at these in a bit more detail now.

Slide courtesy of Rafik Salama

R basics: dataframes

You can make such dataframes in Excel!

Explanatory variables

Response variable

Field.Name	Area	Slope	Vegetation	Soil.pH	Damp	Worm.density
Nashs.Field	3.6	11	Grassland	4.1	F	4
Silwood.Bottom	5.1	2	Arable	5.2	F	7
Nursery.Field	2.8	3	Grassland	4.3	F	2
Rush.Meadow	2.4	5	Meadow	4.9	T	5
Gunness.Thicket	3.8	0	Scrub	4.2	F	6
Oak.Mead	3.1	2	Grassland	3.9	F	2
Church.Field						
Ashurst						
The.Orchard						
Rookery.Slope						
Garden.Wood						
North.Gravel	3.3	1	Grassland	4.1	F	1
South.Gravel	3.7	2	Grassland	4	F	2
Observatory.Ridge	1.8	6	Grassland	3.8	F	0
Pond.Field	4.1	0	Meadow	5	T	6
Water.Meadow	3.9	0	Meadow	4.9	T	8
Cheapside	2.2	8	Scrub	4.7	T	4
Pound.Hill	4.4	2	Arable	4.5	F	5
Gravel.Pit	2.9	1	Grassland	3.5	F	1
Farm.Wood	0.8	10	Scrub	5.1	T	3

All values of the same variable must be in the same column!

Slide courtesy of Rafik Salama

Setting working directory

- R can be used entirely within the console, but it's usually helpful to read in or write out some files at some point
- R looks for files and outputs files to the current working directory
- Checking the current working directory:
  - From the File menu, 'Change directory' allows the user to browse to the directory of interest
  - `> getwd() # prints the path of the current working directory`
  - The `setwd()` function allows a new directory to be specified
  - `> setwd("C:/Users/Anyone/Projects")`

R basics: Reading data from files

- `read.table()` reads a file in table format and creates a data frame from it. Has many optional arguments allowing control over data input: see `help(read.table)` for details
- R ideally expects the file to contain a name for each variable in the data frame in the first row (like column names) and row labels in the first column (usually just numbered) but can be e.g. sample/gene identifiers. Row names must be unique.
- The default settings of read.table then enable the command to be used simply by specifying the name of the file:

```
> df <- read.table("filename.txt")
> read.table(filename, sep="\t", header=T,
row.names=1)
```

R basics: Reading data from files

- Default behaviour is to read numeric items as numeric variables, which is fine, and non-numeric variables (categorical data) as factors – which is sometimes not fine, and something to be aware of.
- It can be switched off by setting the argument 'stringsAsFactors=F' or explicitly giving the classes for each column with the colClasses argument
- A specified number of lines from the input file can be ignored (e.g. with skip=10)
- Common issues with input files include leading/trailing whitespace or extra tabs at the end of lines (can set strip.white=TRUE)
- `read.delim()` works well for files containing gene names/descriptions as long text strings – the presence of certain characters such as commas may affect how R reads in the file and sometimes produces an error with `read.table()`

Exercise: Reading in data

- Visit the webpage associated with the textbook by Michael J Crawley (2005) Statistics: an introduction using R: http://www.imperial.ac.uk/bio/research/crawley/statistics/
- Download the data files (zipped) and open the one called 'worms' in Excel or similar
- Change to the directory where you saved the files and read in this csv file like this:
-
```
> worms <- read.table("worms.csv", sep=",", header=TRUE, row.names=1)
```
- Note the following:
  - File name must be correctly typed, put in quotes and exist where R is looking
  - If it is in a subdirectory, specify the relative path with "./subdirectory/worms.csv" (./ indicates current working directory)
  - The first row is the header, containing the names of the variables
  - If the first column contains the row names (i.e. don't treat Field.Name as variable), set row.names=1.
  - Variable names must be ONE word (no blanks), otherwise R expects more columns of data to exist and will give an error
  - Missing values should be denoted NA (no blanks)

Slide courtesy of Rafik Salama

Exercise: Reading in data

- See what happens if we read in worms.csv without specifying the row.names argument

```
> worms <- read.table("worms.csv", sep=",", header=TRUE)
> head(worms)
```

- Because worms.csv contains a name for each column in the file, R will create a data.frame containing each named column, with the rows simply numbered
- If we were to edit Worms.csv to delete the first column name ('Field.Name') and re-run the above command, the first column of the file would automatically be interpreted as row.names and the resulting data frame would have one fewer column. You can check this with `dim(worms)` in each case

Writing data to file

- Data stored in objects generated in R can easily be written to text files
- To write a subset of the 'worms' dataframe to a new file:

```
> write.table(worms[ ,1:4], "Worms_reduced.txt", sep="\t", quote=F, row.names=T)
```

- Plots can be saved to pdf or other formats (jpeg, png):

```
> pdf("Filename.pdf", onefile=T) # opens pdf file
> plot(x, y, main="Example plot") # make one or more plots
> dev.off() # close the device to finish
```

List of some useful R functions

- Central tendency
  - mean()
  - median()
- Other useful functions
  - sum()
  - prod()
  - length()
  - sort()
  - order()
  - min()
  - max()
  - abs()
- Measures of variation
  - range()
  - quantile() (percentiles)
  - boxplot()
  - var()
  - sd()
  - mad() (median absolute deviation)

Slide courtesy of Rafik Salama