Introduction to R/Bioconductor

MCBIOS-2015 Workshop

Thomas Girke

March 12, 2015
Introduction
- Look and Feel of the R Environment
- R Library Depositories
- Installation
- Getting Around
- Basic Syntax
- Data Types and Subsetting
- Basic Calculations

Important Utilities

Reading and Writing External Data

Some Great R Functions

Graphics Environments
- Base Graphics
- Extended Graphics
Outline

Introduction
- Look and Feel of the R Environment
- R Library Depositories
- Installation
- Getting Around
- Basic Syntax
- Data Types and Subsetting
- Basic Calculations

Important Utilities

Reading and Writing External Data

Some Great R Functions

Graphics Environments
- Base Graphics
- Extended Graphics
Outline

Introduction

Look and Feel of the R Environment
R Library Depositories
Installation
Getting Around
Basic Syntax
Data Types and Subsetting
Basic Calculations

Important Utilities

Reading and Writing External Data

Some Great R Functions

Graphics Environments
Base Graphics
Extended Graphics
Why Using R?

- Complete statistical environment and programming language
- Efficient functions and data structures for data analysis
- Powerful graphics
- Access to fast growing number of analysis packages
- Most widely used language in bioinformatics
- Is standard for data mining and biostatistical analysis
- Technical advantages: free, open-source, available for all OSs

Books & Documentation

- Bioinformatics and Computational Biology Solutions Using R and Bioconductor (Gentleman et al., 2005)
- More on this see "Finding Help" section in UCR Manual
What You’ll Get?

R Gui: OS X

Command-line R: Linux/OS X

R Gui: Windows
RStudio: Alternative Working Environment for R

New integrated development environment (IDE) for R that works well for beginners and developers.

Important shortcuts: Ctrl+Enter (send code), Ctrl+Shift+C (comment/uncomment), Ctrl+1/2 (switch window focus)
Vim-R-Tmux: Command-Line IDE for R

Terminal-based Working Environment for R: Vim-R-Tmux

Introduction to R/Bioconductor

Introduction Look and Feel of the R Environment
Outline

**Introduction**

Look and Feel of the R Environment
R Library Depositories
Installation
Getting Around
Basic Syntax
Data Types and Subsetting
Basic Calculations

**Important Utilities**

Reading and Writing External Data

**Some Great R Functions**

**Graphics Environments**

Base Graphics
Extended Graphics
Package Repositories

- CRAN (>6,000 packages) general data analysis
- Bioconductor (>900 packages) bioscience data analysis
- Omegahat (>90 packages) programming interfaces
Outline

Introduction
- Look and Feel of the R Environment
- R Library Depositories

Installation
- Getting Around
- Basic Syntax
- Data Types and Subsetting
- Basic Calculations

Important Utilities

Reading and Writing External Data

Some Great R Functions

Graphics Environments
- Base Graphics
- Extended Graphics
Installation of R and Add-on Packages

Install R for your operating system from:

http://cran.at.r-project.org

Install RStudio from:

http://www.rstudio.com/ide/download

Installation of CRAN Packages

> install.packages(c("pkg1", "pkg2"))
> install.packages("pkg.zip", repos=NULL)

Installation of Bioconductor Packages

> source("http://www.bioconductor.org/biocLite.R")
> library(BiocInstaller)
> BiocVersion()
> biocLite()
> biocLite(c("pkg1", "pkg2"))

For more details see Bioc Install page and Bioconductor Link
Outline

Introduction
- Look and Feel of the R Environment
- R Library Depositories
- Installation

Getting Around
- Basic Syntax
- Data Types and Subsetting
- Basic Calculations

Important Utilities

Reading and Writing External Data

Some Great R Functions

Graphics Environments
- Base Graphics
- Extended Graphics


**Starting R**

The R GUI versions, including RStudio, under Windows and Mac OS X can be opened by double-clicking their icons. Alternatively, one can start it by typing ’R’ in a terminal (default under Linux).

**Startup/Closing Behavior**

The R environment is controlled by hidden files in the startup directory: .RData, .Rhistory and .Rprofile (optional).

```r
## Closing R
> q()
Save workspace image? [y/n/c]:
```

**Note**

When responding with ’y’, then the entire R workspace will be written to the .RData file which can become very large. Often it is sufficient to just save an analysis protocol in an R source file. This way one can quickly regenerate all data sets and objects.
Getting Around

Create an object with the assignment operator <- (or =)

> object <- ...

List objects in current R session

> ls()

Return content of current working directory

> dir()

Return path of current working directory

> getwd()

Change current working directory

> setwd("/home/user")
Outline

Introduction
  Look and Feel of the R Environment
  R Library Depositories
  Installation
  Getting Around

Basic Syntax
  Data Types and Subsetting
  Basic Calculations

Important Utilities

Reading and Writing External Data

Some Great R Functions

Graphics Environments
  Base Graphics
  Extended Graphics
Basic R Syntax

General R command syntax

> object <- function_name(arguments)
> object <- object[arguments]

Finding help

> ?function_name

Load a library

> library("my_library")

Lists all functions defined by a library

> library(help="my_library")

Load library manual (PDF file)

> vignette("my_library")
Executing R Scripts

Execute an R script from within R

\> \texttt{source("my\_script.R")}

Execute an R script from command-line

\texttt{Rscript my\_script.R}
\texttt{R CMD BATCH my\_script.R}
\texttt{R --slave < my\_script.R}
Outline

Introduction
  Look and Feel of the R Environment
  R Library Depositories
  Installation
  Getting Around
  Basic Syntax

Data Types and Subsetting
  Basic Calculations

Important Utilities

Reading and Writing External Data

Some Great R Functions

Graphics Environments
  Base Graphics
  Extended Graphics
Data Types I

**Numeric data: 1, 2, 3**

```r
> x <- c(1, 2, 3); x
[1] 1 2 3
> is.numeric(x)
[1] TRUE
> as.character(x)
[1] "1" "2" "3"
```

**Character data: "a", "b", "c"**

```r
> x <- c("1", "2", "3"); x
[1] "1" "2" "3"
> is.character(x)
[1] TRUE
> as.numeric(x)
[1] 1 2 3
```
Data Types II

Complex data
> c(1, "b", 3)
[1] "1" "b" "3"

Logical data
> x <- 1:10 < 5
> x

  [1]  TRUE  TRUE  TRUE  TRUE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE
> !x

  [1]  FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
> which(x) # Returns index for the 'TRUE' values in logical vector
[1] 1 2 3 4
Data Objects: Vectors and Factors

**Vectors (1D)**

```r
> myVec <- 1:10; names(myVec) <- letters[1:10]
> myVec[1:5]

a b c d e
1 2 3 4 5

> myVec[c(2,4,6,8)]

b d f h
2 4 6 8

> myVec[c("b", "d", "f")]

b d f
2 4 6
```

**Factors (1D): vectors with grouping information**

```r
> factor(c("dog", "cat", "mouse", "dog", "dog", "cat"))

[1] dog cat mouse dog dog cat
Levels: cat dog mouse
```
Data Objects: Matrices, Data Frames and Arrays

Matrices (2D): two dimensional structures with data of same type

```r
> myMA <- matrix(1:30, 3, 10, byrow = TRUE)
> class(myMA)

[1] "matrix"

> myMA[1:2,]

[1,]  1   2   3   4   5   6   7   8   9  10
[2,] 11  12  13  14  15  16  17  18  19  20

> myMA[1, , drop=FALSE]

[1,]  1   2   3   4   5   6   7   8   9  10
```

Data Frames (2D): two dimensional structures with variable data types

```r
> myDF <- data.frame(Col1=1:10, Col2=10:1)
> myDF[1:2,]

Col1 Col2
1   1  10
2   2   9
```

Arrays: data structure with one, two or more dimensions
Data Objects: Lists and Functions

Lists: containers for any object type

```r
> myL <- list(name="Fred", wife="Mary", no.children=3, child.ages=c(4,7,9))
> myL

$name
[1] "Fred"

$wife
[1] "Mary"

$no.children
[1] 3

$child.ages
[1] 4 7 9

> myL[[4]][1:2]

[1] 4 7
```

Functions: piece of code

```r
> myfct <- function(arg1, arg2, ...) {
+     function_body
+ }
```
General Subsetting Rules

Subsetting by positive or negative index/position numbers

> myVec <- 1:26; names(myVec) <- LETTERS
> myVec[1:4]

A B C D
1 2 3 4

Subsetting by same length logical vectors

> myLog <- myVec > 10
> myVec[myLog]

K L M N O P Q R S T U V W X Y Z
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26

Subsetting by field names

> myVec[c("B", "K", "M")]

B K M
2 11 13

Calling a single column or list component by its name with the $ sign

> iris$Species[1:8]

[1] setosa setosa setosa setosa setosa setosa setosa setosa
Levels: setosa versicolor virginica
Outline

Introduction
  Look and Feel of the R Environment
  R Library Depositories
  Installation
  Getting Around
  Basic Syntax
  Data Types and Subsetting

Basic Calculations

Important Utilities

Reading and Writing External Data

Some Great R Functions

Graphics Environments
  Base Graphics
  Extended Graphics
Basic Operators and Calculations

Comparison operators: ==, ! =, <, >, <=, >=

> 1==1
[1] TRUE

Logical operators: AND: &, OR: |, NOT: !

> x <- 1:10; y <- 10:1
> x > y & x > 5

[1] FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE

Calculations: to look up math functions, see Function Index

> x + y


> sum(x)
[1] 55

> mean(x)
[1] 5.5

> apply(iris[1:6,1:3], 1, mean)

          1          2          3          4          5          6
3.3333333 3.1000000 3.0666667 3.0666667 3.3333333 3.6666667
Outline

Introduction
- Look and Feel of the R Environment
- R Library Depositories
- Installation
- Getting Around
- Basic Syntax
- Data Types and Subsetting
- Basic Calculations

Important Utilities

Reading and Writing External Data

Some Great R Functions

Graphics Environments
- Base Graphics
- Extended Graphics
Combining Objects

The `c` function combines vectors and lists

```r
> c(1, 2, 3)
[1] 1 2 3
> x <- 1:3; y <- 101:103
> c(x, y)
[1] 1 2 3 101 102 103
```

The `cbind` and `rbind` functions can be used to append columns and rows, respectively.

```r
> ma <- cbind(x, y)
> ma
     x y
[1,] 1 101
[2,] 2 102
[3,] 3 103
> rbind(ma, ma)
     x y
[1,] 1 101
[2,] 2 102
[3,] 3 103
[4,] 1 101
[5,] 2 102
[6,] 3 103
```
Accessing Name Slots and Dimensions of Objects

Length and dimension information of objects

> `length(iris$Species)`

[1] 150

> `dim(iris)`

[1] 150 5

Accessing row and column names of 2D objects

> `rownames(iris)[1:8]`

[1] "1" "2" "3" "4" "5" "6" "7" "8"

> `colnames(iris)`


Return name field of vectors and lists

> `names(myVec)`

[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"

> `names(myL)`

[1] "name" "wife" "no.children" "child.ages"
Sorting Objects

The function `sort` returns a vector in ascending or descending order

> `sort(10:1)`

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

The function `order` returns a sorting index for sorting an object

> `sortindex <- order(iris[,1], decreasing = FALSE)`
> `sortindex[1:12]`

```
[1] 14  9 39 43 42  4  7 23 48  3 30 12
```

> `iris[sortindex,][1:2,]`

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
14  4.3     3.0       1.1     0.1  setosa
  9  4.4     2.9       1.4     0.2  setosa
```

> `sortindex <- order(-iris[,1]) # Same as decreasing=TRUE`

Sorting on multiple columns

> `iris[order(iris$Sepal.Length, iris$Sepal.Width),][1:2,]`

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
14   4.3     3.0       1.1     0.1  setosa
  9   4.4     2.9       1.4     0.2  setosa
```
Outline

Introduction
- Look and Feel of the R Environment
- R Library Depositories
- Installation
- Getting Around
- Basic Syntax
- Data Types and Subsetting
- Basic Calculations

Important Utilities

Reading and Writing External Data

Some Great R Functions

Graphics Environments
- Base Graphics
- Extended Graphics
Reading and Writing External Data

Import data from tabular files into R

```r
> myDF <- read.delim("myData.xls", sep="\t")
```

Export data from R to tabular files

```r
> write.table(myDF, file="myfile.xls", sep="\t", quote=FALSE, col.names=NA)
```

Copy and paste (e.g. from Excel) into R

```r
> ## On Windows/Linux systems:
> read.delim("clipboard")
> ## On Mac OS X systems:
> read.delim(pipe("pbpaste"))
```

Copy and paste from R into Excel or other programs

```r
> ## On Windows/Linux systems:
> write.table(iris, "clipboard", sep="\t", col.names=NA, quote=F)
> ## On Mac OS X systems:
> zz <- pipe('pbcopy', 'w')
> write.table(iris, zz, sep="\t", col.names=NA, quote=F)
> close(zz)
```
Task 1  Sort the rows of the iris data frame by its first column and sort its columns alphabetically by column names.

Task 2  Subset the first 12 rows, export the result to a text file and view it in Excel.

Task 3  Change some column titles in Excel and import the result into R.

Structure of iris data set:

```r
> class(iris)
[1] "data.frame"

> dim(iris)
[1] 150 5

> colnames(iris)
```
Outline

Introduction
   Look and Feel of the R Environment
   R Library Depositories
   Installation
   Getting Around
   Basic Syntax
   Data Types and Subsetting
   Basic Calculations

Important Utilities

Reading and Writing External Data

Some Great R Functions

Graphics Environments
   Base Graphics
   Extended Graphics
The `unique()` function to make vector entries unique

```r
> length(iris$Sepal.Length)

[1] 150
```

```r
> length(unique(iris$Sepal.Length))

[1] 35
```

The `table()` function counts the occurrences of entries

```r
> table(iris$Species)

setosa versicolor virginica
    50       50       50
```

The `aggregate()` function computes statistics of data aggregates

```r
> aggregate(iris[,1:4], by=list(iris$Species), FUN=mean, na.rm=TRUE)

    Group.1 Sepal.Length Sepal.Width Petal.Length Petal.Width
1     setosa     5.006     3.428      1.462     0.246
2  versicolor     5.936     2.770      4.260     1.326
3  virginica     6.588     2.974      5.552     2.026
The `%in%` function returns the intersect between two vectors

```r
> month.name %in% c("May", "July")
[1] FALSE FALSE FALSE FALSE TRUE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE
```

The `merge()` function joins two data frames by common field entries, here row names (by.x=0). To obtain only the common rows, change `all=TRUE` to `all=FALSE`. To merge on specific columns, refer to them by their position numbers or their column names.

```r
> frame1 <- iris[sample(1:length(iris[,1]), 30), ]
> frame1[1:2,]

     Sepal.Length Sepal.Width Petal.Length Petal.Width     Species
  129       6.4       2.8       5.6       2.1     virginica
   1       5.1       3.5       1.4       0.2         setosa

> dim(frame1)
[1] 30  5

> my_result <- merge(frame1, iris, by.x = 0, by.y = 0, all = TRUE)
> dim(my_result)
[1] 150 11
Graphics in R

- Powerful environment for visualizing scientific data
- Integrated graphics and statistics infrastructure
- Publication quality graphics
- Fully programmable
- Highly reproducible
- Full \LaTeX \textsuperscript{Link} & Sweave \textsuperscript{Link} support
- Vast number of R packages with graphics utilities
Documentation on Graphics in R

General

- Graphics Task Page
- R Graph Gallery
- R Graphical Manual
- Paul Murrell’s book R (Grid) Graphics

Interactive graphics

- rggobi (GGobi)
- iplots
- Open GL (rgl)
Graphics Environments

Viewing and saving graphics in R
- On-screen graphics
- postscript, pdf, svg
- jpeg, png, wmf, tiff, ...

Four major graphic environments
- Low-level infrastructure
  - R Base Graphics (low- and high-level)
  - grid: Manual Link, Book Link
- High-level infrastructure
  - lattice: Manual Link, Intro Link, Book Link
  - ggplot2: Manual Link, Intro Link, Book Link
Outline

Introduction
  Look and Feel of the R Environment
  R Library Depositories
  Installation
  Getting Around
  Basic Syntax
  Data Types and Subsetting
  Basic Calculations

Important Utilities

Reading and Writing External Data

Some Great R Functions

Graphics Environments
  Base Graphics
  Extended Graphics
Outline

Introduction
Look and Feel of the R Environment
R Library Depositories
Installation
Getting Around
Basic Syntax
Data Types and Subsetting
Basic Calculations

Important Utilities

Reading and Writing External Data

Some Great R Functions

Graphics Environments
Base Graphics
Extended Graphics
Base Graphics: Overview

Important high-level plotting functions

- plot: generic x-y plotting
- barplot: bar plots
- boxplot: box-and-whisker plot
- hist: histograms
- pie: pie charts
- dotchart: cleveland dot plots
- image, heatmap, contour, persp: functions to generate image-like plots
- qnorm, qline, qqplot: distribution comparison plots
- pairs, coplot: display of multivariate data

Help on these functions

- ?myfct
- ?plot
- ?par
Matrices and data frames
Vectors
Named vectors
Scatter Plot: very basic

Sample data set for subsequent plots

```r
> set.seed(1410)
> y <- matrix(runif(30), ncol=3, dimnames=list(letters[1:10], LETTERS[1:3]))

> plot(y[,1], y[,2])
```
Scatter Plot: all pairs

> pairs(y)
Scatter Plot: with labels

```r
> plot(y[,1], y[,2], pch=20, col="red", main="Symbols and Labels")
> text(y[,1]+0.03, y[,2], rownames(y))
```

Symbols and Labels

![Scatter Plot](image)
Scatter Plots: more examples

Print instead of symbols the row names

```r
> plot(y[,1], y[,2], type="n", main="Plot of Labels")
> text(y[,1], y[,2], rownames(y))
```

Usage of important plotting parameters

```r
> grid(5, 5, lwd = 2)
> op <- par(mar=c(8,8,8,8), bg="lightblue")
> plot(y[,1], y[,2], type="p", col="red", cex.lab=1.2, cex.axis=1.2,
+     cex.main=1.2, cex.sub=1, lwd=4, pch=20, xlab="x label",
+     ylab="y label", main="My Main", sub="My Sub")
> par(op)
```

Important arguments

- `mar`: specifies the margin sizes around the plotting area in order: `c(bottom, left, top, right)`
- `col`: color of symbols
- `pch`: type of symbols, samples: `example(points)`
- `lwd`: size of symbols
- `cex.*`: control font sizes
- For details see `?par`
Add a regression line to a plot

> plot(y[,1], y[,2])
> myline <- lm(y[,2]~y[,1]); abline(myline, lwd=2)
> summary(myline)

Same plot as above, but on log scale

> plot(y[,1], y[,2], log="xy")

Add a mathematical expression to a plot

> plot(y[,1], y[,2]); text(y[1,1], y[1,2],
>   expression(sum(frac(1,sqrt(x^2*pi)))), cex=1.3)
Task 1  Generate scatter plot for first two columns in iris data frame and color dots by its Species column.

Task 2  Use the xlim/ylim arguments to set limits on the x- and y-axes so that all data points are restricted to the left bottom quadrant of the plot.

Structure of iris data set:

```r
> class(iris)
[1] "data.frame"

> iris[1:4,]

   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
 1       5.1       3.5         1.4       0.2       setosa
 2       4.9       3.0         1.4       0.2       setosa
 3       4.7       3.2         1.3       0.2       setosa
 4       4.6       3.1         1.5       0.2       setosa

> table(iris$Species)

setosa versicolor virginica
    50        50        50
```
Line Plot: Single Data Set

```r
> plot(y[,1], type="l", lwd=2, col="blue")
```

![Line Plot](image)
Line Plots: Many Data Sets

```r
> split.screen(c(1,1));
[1] 1
> plot(y[,1], ylim=c(0,1), xlab="Measurement", ylab="Intensity", type="l", lwd=2, col=1)
> for(i in 2:length(y[,1])) {
+   screen(1, new=FALSE)
+   plot(y[,i], ylim=c(0,1), type="l", lwd=2, col=i, xaxt="n", yaxt="n", ylab="", xlab="", main="", bty="n")
+ }
> close.screen(all=TRUE)
```

```
0.0 0.2 0.4 0.6 0.8 1.0
2 4 6 8 10
Measurement
Intensity
```

Introduction to R/Bioconductor
Graphics Environments
Base Graphics

Slide 52/62
Bar Plot Basics

```r
> barplot(y[1:4,], ylim=c(0, max(y[1:4,])+0.3), beside=TRUE,
+        legend=letters[1:4])
> text(labels=round(as.vector(as.matrix(y[1:4,])),2), x=seq(1.5, 13, by=1)
+      +sort(rep(c(0,1,2), 4)), y=as.vector(as.matrix(y[1:4,]))+0.04)
```

![Bar plot image](image-url)
Bar Plots with Error Bars

```r
> bar <- barplot(m <- rowMeans(y) * 10, ylim=c(0, 10))
> stdev <- sd(t(y))
> arrows(bar, m, bar, m + stdev, length=0.15, angle = 90)
```
Histograms

> hist(y, freq=TRUE, breaks=10)
Density Plots

\[ \text{plot(density(y), col="red")} \]
Pie Charts

```r
> pie(y[,1], col=rainbow(length(y[,1]), start=0.1, end=0.8), clockwise=TRUE)
> legend("topright", legend=row.names(y), cex=1.3, bty="n", pch=15, pt.cex=1.8,
+ col=rainbow(length(y[,1]), start=0.1, end=0.8), ncol=1)
```
Color Selection Utilities

Default color palette and how to change it

```r
> palette()
```

> palette(rainbow(5, start=0.1, end=0.2))

> palette()

```r
[1] "black"  "red"     "green3" "blue"     "cyan"     "magenta" "yellow"  "gray"
```

```r
> palette("default")
```

The `gray` function allows to select any type of gray shades by providing values from 0 to 1

```r
> gray(seq(0.1, 1, by= 0.2))
```

```r
[1] "#1A1A1A" "#4D4D4D" "#808080" "#B3B3B3" "#E6E6E6"
```

Color gradients with `colorpanel` function from `gplots` library

```r
> library(gplots)
> colorpanel(5, "darkblue", "yellow", "white")
```

Much more on colors in R see Earl Glynn’s color chart [Link]
After the `pdf()` command all graphs are redirected to file `test.pdf`. Works for all common formats similarly: jpeg, png, ps, tiff, ...

```
> pdf("test.pdf"); plot(1:10, 1:10); dev.off()
```

Generates Scalable Vector Graphics (SVG) files that can be edited in vector graphics programs, such as Inkscape.

```
> svg("test.svg"); plot(1:10, 1:10); dev.off()
```
Exercise 3: Bar Plots

**Task 1** Calculate the mean values for the Species components of the first four columns in the iris data set. Organize the results in a matrix where the row names are the unique values from the iris Species column and the column names are the same as in the first four iris columns.

**Task 2** Generate two bar plots: one with stacked bars and one with horizontally arranged bars.

Structure of iris data set:

```r
> class(iris)
[1] "data.frame"

> iris[1:4,]

    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
   1        5.1       3.5       1.4      0.2  setosa
   2        4.9       3.0       1.4      0.2  setosa
   3        4.7       3.2       1.3      0.2  setosa
   4        4.6       3.1       1.5      0.2  setosa

> table(iris$Species)

    setosa versicolor virginica
       50        50        50
```
Outline

Introduction

Look and Feel of the R Environment
R Library Depositories
Installation
Getting Around
Basic Syntax
Data Types and Subsetting
Basic Calculations

Important Utilities

Reading and Writing External Data

Some Great R Functions

Graphics Environments

Base Graphics
Extended Graphics
High-level Graphics: \textit{ggplot2} and \textit{lattice}

- See here

[Link]