Introduction to R

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

R

- Language and environment for statistics and visualisation
- Great and easy to make plots
- Extensible via Libraries:
  - CRAN (The Comprehensive R Archive Network) - 5968 Packages
    (WS2013/14: 4118, WS2011/12: 3087, WS 2010/11: 2561, SS 2010: 2326, SS 2009: 1796)
  - Bioconductor - more than 700 Software Packages many for bioinformatics: Microarrays, High Throughput Assays, sequence data, annotations (KEGG, GO, NCBI)

The Golden Age of Data Mining

The New York Times

To some people R is just the 18th letter of the alphabet . . . or what pirates in movies say. R is also the name of a popular programming language used by a growing number of data analysts inside corporations and academia. It is becoming their lingua franca partly because **data mining has entered a golden age**, *whether being used to set ad prices, find new drugs more quickly or fine-tune financial models. Companies as diverse as Google, Pfizer, Merck, Bank of America, the InterContinental Hotels Group and Shell use it.*

www.nytimes.com/2009/01/07/technology/business-computing/07program.html

To start / end R: on the console

```
user@host:~$ R

R version 3.2.2 (2015-08-14) -- "Fire Safety"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.

...

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> q()
Save workspace image? [y/n/c]: n
user@host:~$
```

To start / end R: on the console

```
user@host:~$ R

R version 3.2.2 (2015-08-14) -- "Fire Safety"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.

...

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> q()
Save workspace image? [y/n/c]: n
user@host:~$
```

To start / end R: using an IDE

Recommended:
RStudio
or
on a Mac: R For Mac OS X GUI

Variables

```
> a <- 2
> a
[1] 2
> cat("a*5 is", a * 5, "\n")
a*5 is 10
> b <- a
> b
[1] 2
> b == a
[1] TRUE
> if (b == a) {
+     cat("Yes!\n")
+ }
Yes!
```

Variables

```
> a <- 2
> a
[1] 2
> cat("a*5 is", a * 5, "\n")
a*5 is 10
> b <- a
> b
[1] 2
> b == a
[1] TRUE
> if (b == a) {
+     cat("Yes!\n")
+ }
Yes!
```

Functions

```
> fib <- function(x) {
+     if (x == 1 || x == 0) {
+         return(x)
+     }
+     else {
+         return(fib(x - 1) + fib(x - 2))
+     }
+ }
> fib(7)
[1] 13
> ls()
[1] "a"   "b"   "fib"
> rm(a)
> ls()
[1] "b"   "fib"
```

Functions

```
> fib <- function(x) {
+     if (x == 1 || x == 0) {
+         return(x)
+     }
+     else {
+         return(fib(x - 1) + fib(x - 2))
+     }
+ }
> fib(7)
[1] 13
> ls()
[1] "a"   "b"   "fib"
> rm(a)
> ls()
[1] "b"   "fib"
```

Functions

```
> fib <- function(x) {
+     if (x == 1 || x == 0) {
+         return(x)
+     }
+     else {
+         return(fib(x - 1) + fib(x - 2))
+     }
+ }
> fib(7)
[1] 13
> ls()
[1] "a"   "b"   "fib"
> rm(a)
> ls()
[1] "b"   "fib"
```

Data types

- Simple Data types: logical, numeric, character
- Data structures: depends on dimension and content

Dimension	uniform	non-uniform
1D	vector	list
2D	matrix	data.frame

Data types

Convert data types:

```
> a <- c(T, F)
> a
[1]  TRUE FALSE

> mode(a)
[1] "logical"

> a <- as.numeric(a)
> a
[1] 1 0

> a <- as.character(a)
> a
[1] "1" "0"
```

Data types

Convert data types:

```
> a <- c(T, F)
> a
[1]  TRUE FALSE

> mode(a)
[1] "logical"

> a <- as.numeric(a)
> a
[1] 1 0

> a <- as.character(a)
> a
[1] "1" "0"
```

Vectors and Indexing

```
> v <- 1:12
> v
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
> v[1]
[1] 1
> v[2:4]
[1] 2 3 4
> v[-7:-9]
 [1]  1  2  3  4  5  6 10 11 12
> v <- c(1, 2, 5)
> v[c(TRUE, T, FALSE)]
[1] 1 2
```

Vectors and Indexing

```
> v <- 1:12
> v
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
> v[1]
[1] 1
> v[2:4]
[1] 2 3 4
> v[-7:-9]
 [1]  1  2  3  4  5  6 10 11 12
> v <- c(1, 2, 5)
> v[c(TRUE, T, FALSE)]
[1] 1 2
```

Vectors and Indexing

```
> v <- 1:12
> v
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
> v[1]
[1] 1
> v[2:4]
[1] 2 3 4
> v[-7:-9]
 [1]  1  2  3  4  5  6 10 11 12
> v <- c(1, 2, 5)
> v[c(TRUE, T, FALSE)]
[1] 1 2
```

Vectors and Indexing

```
> v <- 1:12
> v
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
> v[1]
[1] 1
> v[2:4]
[1] 2 3 4
> v[-7:-9]
 [1]  1  2  3  4  5  6 10 11 12
> v <- c(1, 2, 5)
> v[c(TRUE, T, FALSE)]
[1] 1 2
```

Vectors and Indexing

```
> v <- 1:12
> v
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
> v[1]
[1] 1
> v[2:4]
[1] 2 3 4
> v[-7:-9]
 [1]  1  2  3  4  5  6 10 11 12
> v <- c(1, 2, 5)
> v[c(TRUE, T, FALSE)]
[1] 1 2
```

Comparisons

```
> v <- 1:5
> v == 3
[1] FALSE FALSE  TRUE FALSE FALSE
> all(v > 2)
[1] FALSE
> any(v < 3)
[1] TRUE
> sum(v > 2.5)
[1] 3
> which(v > 2.5)
[1] 3 4 5
```

Comparisons

```
> v <- 1:5
> v == 3
[1] FALSE FALSE  TRUE FALSE FALSE
> all(v > 2)
[1] FALSE
> any(v < 3)
[1] TRUE
> sum(v > 2.5)
[1] 3
> which(v > 2.5)
[1] 3 4 5
```

Comparisons

```
> v <- 1:5
> v == 3
[1] FALSE FALSE  TRUE FALSE FALSE
> all(v > 2)
[1] FALSE
> any(v < 3)
[1] TRUE
> sum(v > 2.5)
[1] 3
> which(v > 2.5)
[1] 3 4 5
```

Comparisons

```
> v <- 1:5
> v == 3
[1] FALSE FALSE  TRUE FALSE FALSE
> all(v > 2)
[1] FALSE
> any(v < 3)
[1] TRUE
> sum(v > 2.5)
[1] 3
> which(v > 2.5)
[1] 3 4 5
```

Comparisons

```
> v <- 1:5
> v == 3
[1] FALSE FALSE  TRUE FALSE FALSE
> all(v > 2)
[1] FALSE
> any(v < 3)
[1] TRUE
> sum(v > 2.5)
[1] 3
> which(v > 2.5)
[1] 3 4 5
```

Comparisons

```
> v <- 1:5
> v == 3
[1] FALSE FALSE  TRUE FALSE FALSE
> all(v > 2)
[1] FALSE
> any(v < 3)
[1] TRUE
> sum(v > 2.5)
[1] 3
> which(v > 2.5)
[1] 3 4 5
```

Combining

Adding elements:

```
> v <- 1:5
> v[6] <- 12
> v
```

```
[1]  1  2  3  4  5 12
```

`c(...)` combines objects

```
> w <- c(v, v)
> length(w)
```

```
[1] 12
```

```
> sum(w)
```

```
[1] 54
```

Lists (1)

Lists can contain data of different types

```
> L <- list(2, "foo", 1:7)

> length(L)

[1] 3

> L
[[1]]
[1] 2


[[2]]
[1] "foo"


[[3]]
[1] 1 2 3 4 5 6 7
```

Lists (1)

Lists can contain data of different types

```
> L <- list(2, "foo", 1:7)
```

```
> length(L)
```

```
[1] 3
```

```
> L
[[1]]
[1] 2


[[2]]
[1] "foo"


[[3]]
[1] 1 2 3 4 5 6 7
```

Lists (1)

Lists can contain data of different types

```
> L <- list(2, "foo", 1:7)

> length(L)

[1] 3

> L
[[1]]
[1] 2

[[2]]
[1] "foo"

[[3]]
[1] 1 2 3 4 5 6 7
```

Lists (2)

Access to individual elements of lists:

```
> L[[2]]
[1] "foo"
> L[1:2]
[[1]]
[1] 2

[[2]]
[1] "foo"
> K <- list(apple = 42, egg = 23)
> K$apple
[1] 42
> K$egg
[1] 23
```

Lists (2)

Access to individual elements of lists:

```
> L[[2]]
[1] "foo"
> L[1:2]
[[1]]
[1] 2

[[2]]
[1] "foo"
> K <- list(apple = 42, egg = 23)
> K$apple
[1] 42
> K$egg
[1] 23
```

Lists (3)

Convert lists into vectors with `unlist`:

```
> a <- list(1, 2, 3)
> a
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

> a <- unlist(a)
> a
[1] 1 2 3
```

Matrices

```
> m <- matrix(1:8, 2, 4)
> m
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
> dim(m)
[1] 2 4
> m[2, 3]
[1] 6
> m[1, ]
[1] 1 3 5 7
> m[, 3]
[1] 5 6
```

Matrices

```
> m <- matrix(1:8, 2, 4)
> m
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
> dim(m)
[1] 2 4
> m[2, 3]
[1] 6
> m[1, ]
[1] 1 3 5 7
> m[, 3]
[1] 5 6
```

Matrices

```
> m <- matrix(1:8, 2, 4)
> m
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
> dim(m)
[1] 2 4
> m[2, 3]
[1] 6
> m[1, ]
[1] 1 3 5 7
> m[, 3]
[1] 5 6
```

Matrices

```
> m <- matrix(1:8, 2, 4)
> m
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
> dim(m)
[1] 2 4
> m[2, 3]
[1] 6
> m[1, ]
[1] 1 3 5 7
> m[, 3]
[1] 5 6
```

Matrices

```
> m <- matrix(1:8, 2, 4)
> m
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
> dim(m)
[1] 2 4
> m[2, 3]
[1] 6
> m[1, ]
[1] 1 3 5 7
> m[, 3]
[1] 5 6
```

Matrices

```
> m <- matrix(1:8, 2, 4)
> m
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
> dim(m)
[1] 2 4
> m[2, 3]
[1] 6
> m[1, ]
[1] 1 3 5 7
> m[, 3]
[1] 5 6
```

Matrices

```
> m <- matrix(1:8, 2, 4)
> m
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
> dim(m)
[1] 2 4
> m[2, 3]
[1] 6
> m[1, ]
[1] 1 3 5 7
> m[, 3]
[1] 5 6
```

Matrices

```
> m <- matrix(1:8, 2, 4)
> m
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
> dim(m)
[1] 2 4
> m[2, 3]
[1] 6
> m[1, ]
[1] 1 3 5 7
> m[, 3]
[1] 5 6
```

Computing with matrices

```
> m + 1
```

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

```
> t(m)
```

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

```
> m %*% t(m)
```

```
     [,1] [,2]
[1,]   84  100
[2,]  100  120
```

Computing with matrices

```
> m + 1
```

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

```
> t(m)
```

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

```
> m %*% t(m)
```

```
     [,1] [,2]
[1,]   84  100
[2,]  100  120
```

Computing with matrices

```
> m + 1
```

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

```
> t(m)
```

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

```
> m %*% t(m)
```

```
     [,1] [,2]
[1,]   84  100
[2,]  100  120
```

Computing with matrices

```
> m + 1
```

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

```
> t(m)
```

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

```
> m %*% t(m)
```

```
     [,1] [,2]
[1,]   84  100
[2,]  100  120
```

Computing with matrices

```
> m + 1
```

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

```
> t(m)
```

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

```
> m %*% t(m)
```

```
     [,1] [,2]
[1,]   84  100
[2,]  100  120
```

Computing with matrices

```
> m + 1
```

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

```
> t(m)
```

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

```
> m %*% t(m)
```

```
     [,1] [,2]
[1,]   84  100
[2,]  100  120
```

Loops and apply

Loops:

```
> for (i in 1:nrow(m)) {
+     print(mean(m[i, ]))
+ }
```

```
[1] 4
[1] 5
```

apply ist often much faster:

```
> apply(m, 1, mean)
```

```
[1] 4 5
```

Loops and apply

Loops:

```
> for (i in 1:nrow(m)) {
+     print(mean(m[i, ]))
+ }
```

```
[1] 4
[1] 5
```

apply ist often much faster:

```
> apply(m, 1, mean)
```

```
[1] 4 5
```

Apply

`apply` applies a funktion to each row/columns of a matrix

- Syntax: `apply(matrix, margin, function);`
- Margin: Rows (1) or Columns (2)
- Result is a vector with nrow/ncol elements

Beispiel:

```
> m <- matrix(rnorm(2000), 200, 10)
> length(apply(m, 1, mean))
[1] 200
> length(apply(m, 2, sd))
[1] 10
```

Read and save data in R

R can read almost any type of data from text files:

```
> write.table(d, file = "Students.csv")
> read.table(file = "Students.csv")
```

```
  name age
1  Bob  24
2 Mary  22
```

Result is a data frame.

Change working directory:

```
> setwd("/home/user/tmp/")
```

Get actual working directory:

```
> getwd()
```

```
[1] "home/user/tmp"
```

Read and save data in R

R can read almost any type of data from text files:

```
> write.table(d, file = "Students.csv")
> read.table(file = "Students.csv")
```

```
  name age
1  Bob  24
2 Mary  22
```

Result is a data frame.

Change working directory:

```
> setwd("/home/user/tmp/")
```

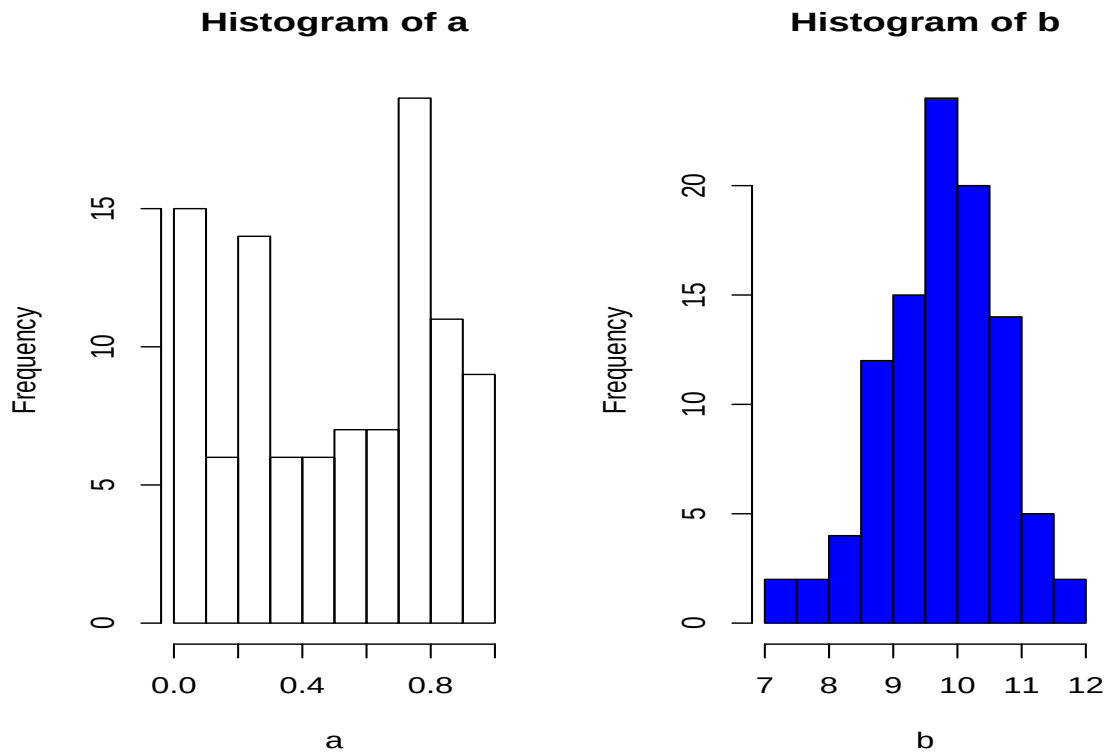
Get actual working directory:

```
> getwd()
```

```
[1] "home/user/tmp"
```

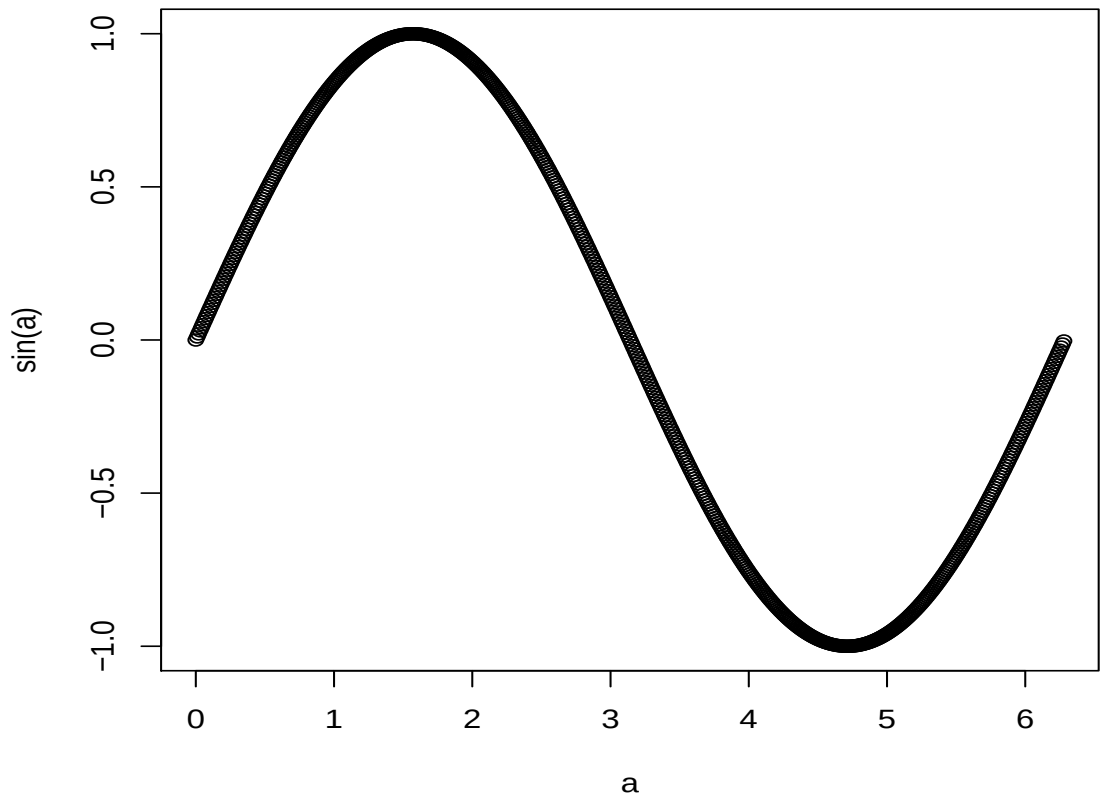
Random numbers

```
> a <- runif(100)
> b <- rnorm(100, mean = 10)
> par(mfrow = c(1, 2))
> hist(a)
> hist(b, co = "BLUE", main = "Histogram of b")
```



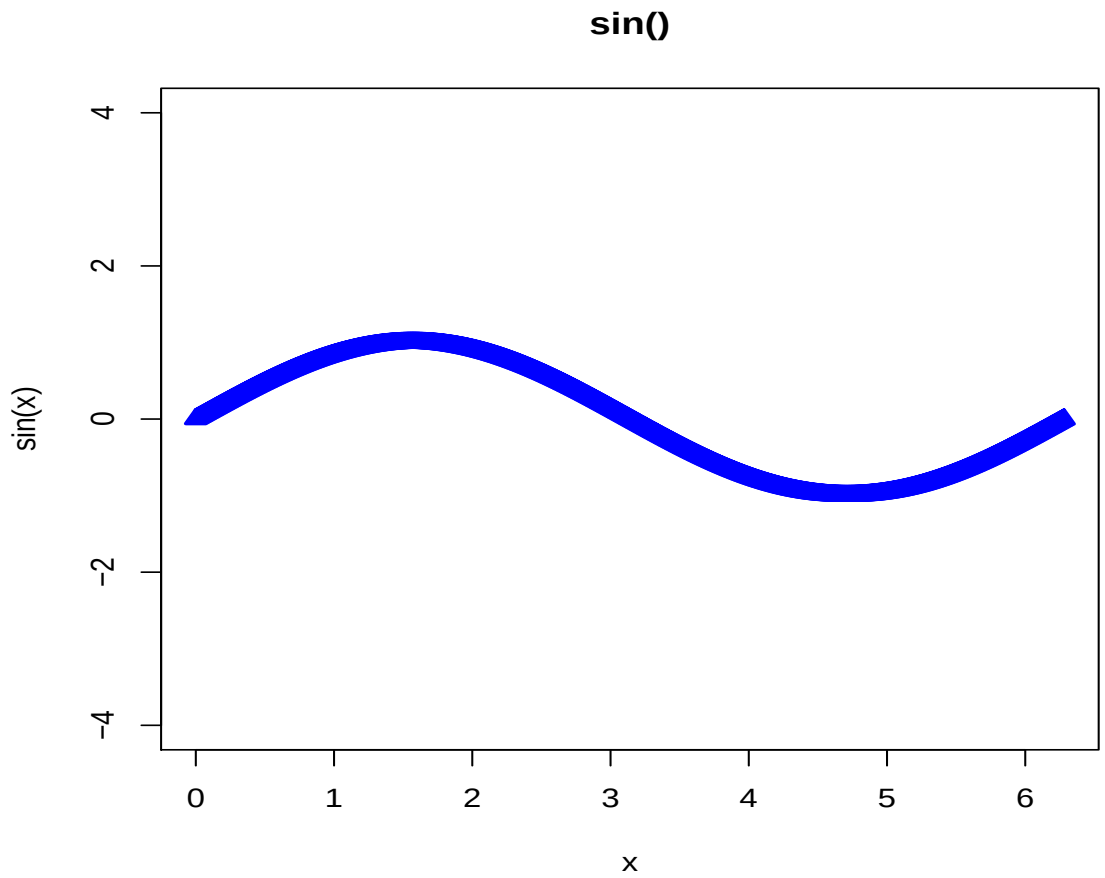
Plotting

```
> a <- seq(0, 2 * pi, 0.01)
> plot(a, sin(a))
```



Plotting (2)

```
> plot(a, sin(a), type = "b", pch = 2, col = "blue",
+      lwd = 2, main = "sin()",
+      ylim = c(-4, 4), ylab = "sin(x)", xlab = "x")
```



Plotting (3)

Save new plot into a file:

```
> png("test.png")
> plot(1:10, 10:1)
> dev.off()
null device
          1
```

Save open Plot into a file:

```
> plot(1:10, 10:1)
> dev.copy(png,"test.png")
png
  3
> dev.off()
```

Bioconductor

Load/install packages:

```
source("http://bioconductor.org/biocLite.R")
```
Where one finds Bioconductor

```
biocLite("limma")
```
install package `limma`

```
library(limma)
```
load installed package `limma`

Help!

- Online help: `help(foo)`
- short form `?foo`
- Full text search: `help.search("keyword")`
- Regexp search: `apropos("foo")`

CRAN / Bioconductor-Pakete offer documentation

- Web - ask uncle Google
- `vignette("foo")` offers extensive help for package here "foo"
- `vignette()` lists help topics.

Help!

- Online help: `help(foo)`
- short form `?foo`
- Full text search: `help.search("keyword")`
- Regexp search: `apropos("foo")`

CRAN / Bioconductor-Pakete offer documentation

- Web - ask uncle Google
- `vignette("foo")` offers extensive help for package here "foo"
- `vignette()` lists help topics.