Crash n' Survival Course with R

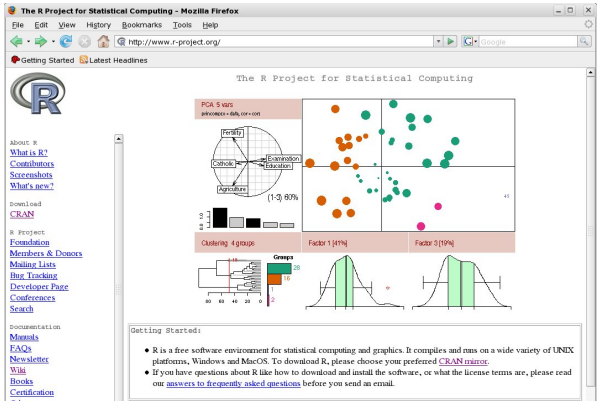
Anne Kupczok

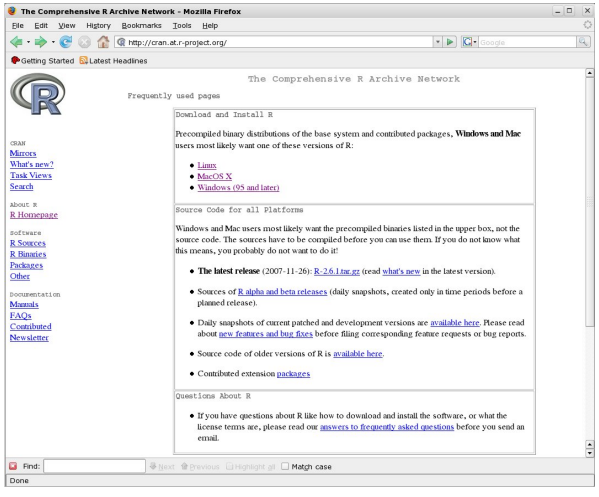
MFPL - CIBIV
Center for Integrative Bioinformatics in Vienna

February 9, 2009

Your biggest friend and partner in statistics

Installation of R, starting and additional libraries





- R Webpage http://www.r-project.org/
- There is code and precompiled binaries
- Start with the command `R`
- Additional libraries which are installed are loaded with `library(name)`
- A usefull interface is the R commander which is started with `library(Rcmdr)`, conatins a script and an output window

Using R as calculator: Basic Operations

`+ , -`

```
> 5-2 # addition, subtraction
[1] 3
```

`* , /`

```
> 5*2 # multiplication, division
[1] 10
```

`^ or **`

```
> 5**2 # power
[1] 25
```

`%/%`

```
> 5%/%2 # division
[1] 2
```

`%%`

```
> 5%%2 # modulo
[1] 1
```

Basic arithmetic functions and values

predefined arithmetic functions:

max(), min()
abs()
sqrt()
round, floor(), ceiling()
sum(), prod()
log(), log10(), log2()
exp()
sin(), cos(), tan(), asin(), acos(), atan()

Basic arithmetic functions and values

predefined arithmetic functions:

max(), min()
abs()
sqrt()
round, floor(), ceiling()
sum(), prod()
log(), log10(), log2()
exp()
sin(), cos(), tan(), asin(), acos(), atan()

Constant Values:

pi	π	pi
Inf, -Inf	infinity	1/0
NaN	Not a Number	0/0
NA	Not Available	'NA'
NULL	Not defined	

Assigning values to variables

We do not need to declare the datatype of a variable in R.

The simplest datatype in R is a scalar.

- variables can be used to represent values:

```
> x <- 15
> x
[1] 15
```

- values can also be assigned by:

```
> a = 15
> a
[1] 15
```

- Logical values:

```
> a == 15
[1] TRUE
```

A Vector is a array of values

construct a vector with the "concatenate function" c()

Scalar vector:

```
> v <- c(15, 4, 67, 5, 9)
> v
[1] 15 4 67 5 9
```

A Vector is a array of values

construct a vector with the "concatenate function" c()

Scalar vector:

```
> v <- c(15, 4, 67, 5, 9)
> v
[1] 15 4 67 5 9
```

Logical vector:

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

A Vector is a array of values

construct a vector with the "concatenate function" c()

Scalar vector:

```
> v <- c(15, 4, 67, 5, 9)
> v
[1] 15 4 67 5 9
```

Logical vector:

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

Character vector:

```
> v <- c("Hund", "Katze", "Maus", "Mensch")
> v
[1] "Hund" "Katze" "Maus" "Mensch"
```

Selections on vectors

- Single value from vector:

```
> x <- c(4, 5, 8, 23, 12)
> x[2]
[1] 5
```

Selections on vectors

- Single value from vector:

```
> x <- c(4, 5, 8, 23, 12)
> x[2]
[1] 5
```

- Multiple values from vector:

```
> x[c(2,4)]
[1] 5 23
```

Selections on vectors

- Single value from vector:

```
> x <- c(4, 5, 8, 23, 12)
> x[2]
[1] 5
```

- Multiple values from vector:

```
> x[c(2,4)]
[1] 5 23
```

- Subsequent values from vector:

```
> x[1:3]
[1] 4 5 8
```

Selections on vectors

- Single value from vector:

```
> x <- c(4, 5, 8, 23, 12)
> x[2]
[1] 5
```

- Multiple values from vector:

```
> x[c(2,4)]
[1] 5 23
```

- Subsequent values from vector:

```
> x[1:3]
[1] 4 5 8
```

- Logical selection (boolean vectors of same length):

```
> x>7
[1] FALSE FALSE  TRUE  TRUE  TRUE
> x[x>7]
[1]  8 23 12
```

Operations on vectors

- arithmetic operations on vectors:

```
> x <- c(4, 5, 8, 23, 12)
> x*2
[1]  8 10 16 46 24
```

Operations on vectors

- arithmetic operations on vectors:

```
> x <- c(4, 5, 8, 23, 12)
> x*2
[1]  8 10 16 46 24
```

- Vectors need to have the same length!

```
> x + x
[1]  8 10 16 46 24
> x + x[1:4]
[1]  8 10 16 46 16
Warning message: longer object length is not a
multiple of shorter object length in: x + x[1:4]
```

Operations on vectors

- arithmetic operations on vectors:

```
> x <- c(4, 5, 8, 23, 12)
> x*2
[1]  8 10 16 46 24
```

- Vectors need to have the same length!

```
> x + x
[1]  8 10 16 46 24
> x + x[1:4]
[1]  8 10 16 46 16
Warning message: longer object length is not a
multiple of shorter object length in: x + x[1:4]
```

- Test whether a vector contains an element

```
> c(5,6) %in% x
[1]  TRUE FALSE
```

Some functions on vectors

- Getting the length of a vector

```
> x <- c(4, 5, 8, 23, 12)
> length(x)
[1] 5
```

Some functions on vectors

- Getting the length of a vector

```
> x <- c(4, 5, 8, 23, 12)
> length(x)
[1] 5
```

- Getting and changing the names of a vector

```
> names(x)=c("E1","E2","E3","E4","E5")
> x
E1 E2 E3 E4 E5
 4  5  8 23 12
> x["E3"]
E3
 8
```

Some functions on vectors

- Getting the length of a vector

```
> x <- c(4, 5, 8, 23, 12)
> length(x)
[1] 5
```

- Getting and changing the names of a vector

```
> names(x)=c("E1","E2","E3","E4","E5")
> x
E1 E2 E3 E4 E5
 4  5  8 23 12
> x["E3"]
E3
 8
```

- Getting the order of a vector

```
> o=order(x)
> o
[1] 1 2 3 5 4
> sort(x) #same as x[o]
[1]  4  5  8 12 23
```

Matrices are represented as vectors with two dimensions

We can set or change dimensions with the function dim

```
> M <- 1:6
> dim(M) <- c(2,3)
> M
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Matrices are represented as vectors with two dimensions

We can set or change dimensions with the function dim

```
> M <- 1:6
> dim(M) <- c(2,3)
> M
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Matrices can be created using the matrix function

```
> N= matrix(1:6,nrow=2,byrow=T)
> N
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> dim(N)
[1] 2 3
```

(here the Matrix will be filled in a rowwise fashion)

Useful functions that operate on matrices

Indexing

```
> MAT<-matrix(5:12,c(4,2),byrow=T)
> MAT[3]
[1] 7
> MAT[,1]
[1]  5  7  9 11
> MAT[1,]
[1] 5 6
> MAT[3,2]
[1] 10
```

```
> MAT
     [,1] [,2]
[1,]    5    6
[2,]    7    8
[3,]    9   10
[4,]   11   12
```

Useful functions that operate on matrices

Indexing

```
> MAT<-matrix(5:12,c(4,2),byrow=T)
> MAT[3]
[1] 7
> MAT[,1]
[1]  5  7  9 11
> MAT[1,]
[1] 5 6
> MAT[3,2]
[1] 10
```

```
> MAT
     [,1] [,2]
[1,]    5    6
[2,]    7    8
[3,]    9   10
[4,]   11   12
```

assign rownames and colnames

```
> rownames(M) <- c("X","Y")
> colnames(M) <- c("A","B","C")
> M
  A B C
X 1 3 5
Y 2 4 6
> M["X",]
A B C
1 3 5
```

Factors: storing categorical variables

A factor consists of a vector of integers and a character vector

```
> pain <- c(0,3,2,2,1)
> fpain <- factor(pain,levels=0:3)
> levels(fpain) <- c("none","mild","medium","severe")
```

Factors: storing categorical variables

A factor consists of a vector of integers and a character vector

```
> pain <- c(0,3,2,2,1)
> fpain <- factor(pain,levels=0:3)
> levels(fpain) <- c("none","mild","medium","severe")
```

The level argument assures the ordering of our categories
(otherwise alphabetical ordering)

```
> fpain
[1] none   severe medium medium mild
Levels: none mild medium severe
> as.numeric(fpain)
[1] 1 4 3 3 2
```

Factors: storing categorical variables

A factor consists of a vector of integers and a character vector

```
> pain <- c(0,3,2,2,1)
> fpain <- factor(pain,levels=0:3)
> levels(fpain) <- c("none","mild","medium","severe")
```

The level argument assures the ordering of our categories
(otherwise alphabetical ordering)

```
> fpain
[1] none   severe medium medium mild
Levels: none mild medium severe
> as.numeric(fpain)
[1] 1 4 3 3 2
```

Count the categories (can also be applied to normal vectors):

```
> table(fpain)
fpain
  none   mild medium severe
     1      1      2      1
```

Data frames: Lists of vectors with the same length

Creating a data frame object

```
> vector1 <- c("Hund","Katze","Maus","Schnabeltier")
> vector2 <- c(3,5,7,2)
> dframe<-data.frame(Spec=vector1,Num=vector2)
> dframe
          Spec Num
1         Hund   3
2        Katze   5
3         Maus   7
4 Schnabeltier   2
```

Data frames: Selection

- get the names of a object

```
> names(dframe)
[1] "Spec" "Num"
```

Data frames: Selection

- get the names of a object

```
> names(dframe)
[1] "Spec" "Num"
```

- $ gives the column vectors

```
> dframe$Num #equals  dframe[,2] and dframe[,"Num"]
[1] 3 5 7 2
```

Data frames: Selection

- get the names of a object

```
> names(dframe)
[1] "Spec" "Num"
```

- $ gives the column vectors

```
> dframe$Num #equals  dframe[,2] and dframe[,"Num"]
[1] 3 5 7 2
```

- can be indexed like a matrix

```
> dframe$Num[1] # equals dframe[1,2]
[1] 7
```

Lists: composite collection of objects

- creating object using list()

```
> lobject<-list(c(1,2,2,2),age=c(98,21,56))
> lobject
[[1]]
[1] 1 2 2 2

$age
[1] 98 21 56
```

- selection of particular values (Indexing)

```
> lobject[[1]] #is a vector
[1] 1 2 2 2
> lobject$age #also a vector
[1] 98 21 56
> lobject[1] #is still a list
[[1]]
[1] 1 2 2 2
```

Testing and converting variable types

- Test a special type

```
> is.list(lobject)
[1] TRUE
> is.list(lobject[[1]])
[1] FALSE
> is.vector(lobject[[1]])
[1] TRUE
> is.na(c(1,0/0))
[1] FALSE  TRUE
```

Testing and converting variable types

- Test a special type

```
> is.list(lobject)
[1] TRUE
> is.list(lobject[[1]])
[1] FALSE
> is.vector(lobject[[1]])
[1] TRUE
> is.na(c(1,0/0))
[1] FALSE  TRUE
```

- Convert to a special type

```
> a <- as.logical(c(3,2,0,1))
> a
[1]  TRUE  TRUE FALSE  TRUE
> as.integer(a)
[1] 1 1 0 1
```

Read and write data

Simple example how to read a table into a data frame object:

```
> dataFrame1 <- read.table("data.txt",header=T)
```

You can also write a data.frame into a file:

```
> write.table(data,filename)
```

Read and write data

Simple example how to read a table into a data frame object:

```
> dataFrame1 <- read.table("data.txt",header=T)
```

You can also write a data.frame into a file:

```
> write.table(data,filename)
```

Write R-Objects:

```
> #save all the variables in the workspace to a file:
> save(file=filename)
> #save only the variables MAT and v to a file:
> save(MAT,v,file=filename2)
> load(filename) #load the data in another R session
> #get the names of variables in the workspace:
> ls()
```

Read and write data

Simple example how to read a table into a data frame object:

```
> dataFrame1 <- read.table("data.txt",header=T)
```

You can also write a data.frame into a file:

```
> write.table(data,filename)
```

Write R-Objects:

```
> #save all the variables in the workspace to a file:
> save(file=filename)
> #save only the variables MAT and v to a file:
> save(MAT,v,file=filename2)
> load(filename) #load the data in another R session
> #get the names of variables in the workspace:
> ls()
```

Load a script with R-commands:

```
> source(script)
```

Read and write data

Simple example how to read a table into a data frame object:

```
> dataFrame1 <- read.table("data.txt",header=T)
```

You can also write a data.frame into a file:

```
> write.table(data,filename)
```

Write R-Objects:

```
> #save all the variables in the workspace to a file:
> save(file=filename)
> #save only the variables MAT and v to a file:
> save(MAT,v,file=filename2)
> load(filename) #load the data in another R session
> #get the names of variables in the workspace:
> ls()
```

Load a script with R-commands:

```
> source(script)
```

Get and set the working directory of R:

```
> getwd()
> setwd(dirstring)
```

R as a programming language

- Loops: for, while and repeat

```
> v=c(1,2,3,4,5)
> for (i in 1:5){
+ print (sum(v[1:i]))
+ }
[1] 1
[1] 3
[1] 6
[1] 10
[1] 15
```

- There is a much faster way

```
> sapply(1:5,function(i) sum(v[1:i]))
[1]  1  3  6 10 15
```

R as a programming language

- Loops: for, while and repeat

```
> v=c(1,2,3,4,5)
> for (i in 1:5){
+ print (sum(v[1:i]))
+ }
[1] 1
[1] 3
[1] 6
[1] 10
[1] 15
```

- If and else

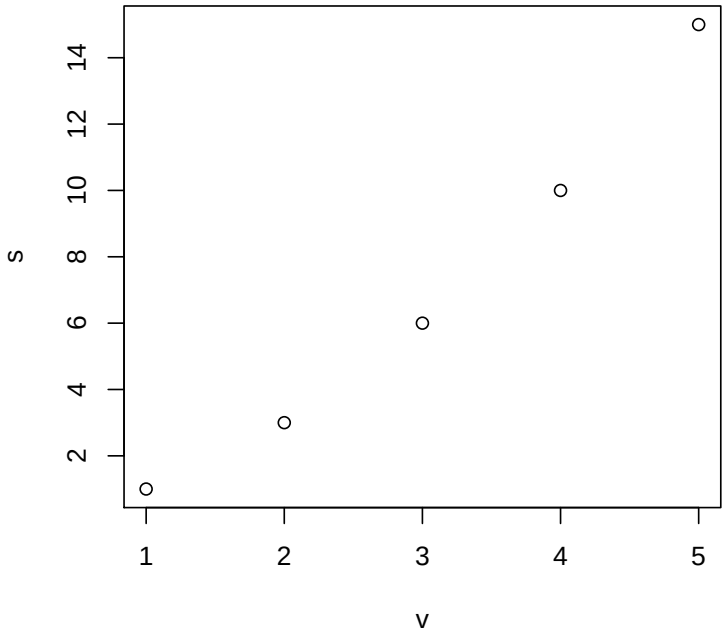
```
> if (v[3]==3){
+ v[2]=3
+ }
> v
[1] 1 3 3 4 5
```

- There is a much faster way

```
> sapply(1:5,function(i) sum(v[1:i]))
[1]  1  3  6 10 15
```

Plotting with R

```
> v = c(1,2,3,4,5)
> s = sapply(1:5,function(i)
+ sum(v[1:i]))
> plot(v,s)
```

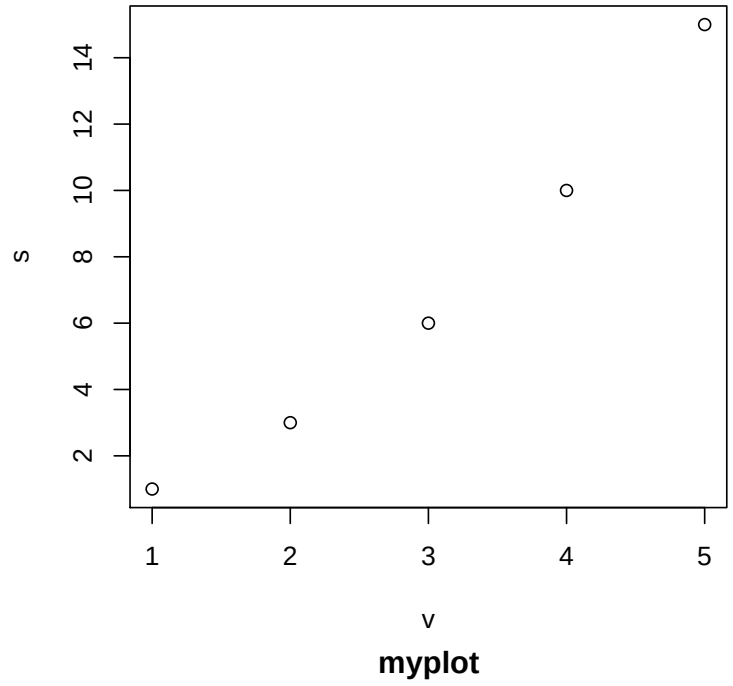


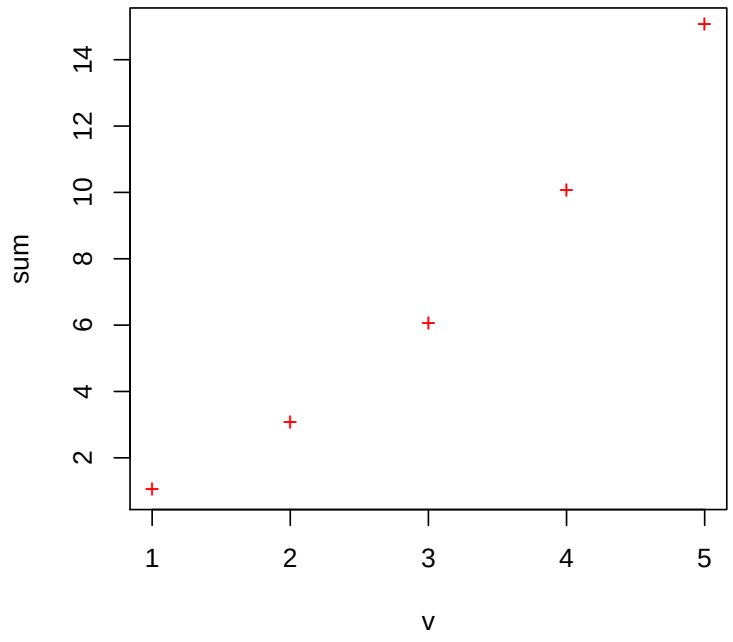
Plotting with R

```
> v = c(1,2,3,4,5)
> s = sapply(1:5,function(i)
+ sum(v[1:i]))
> plot(v,s)
```

There are many parameters to modify the plot, e.g.

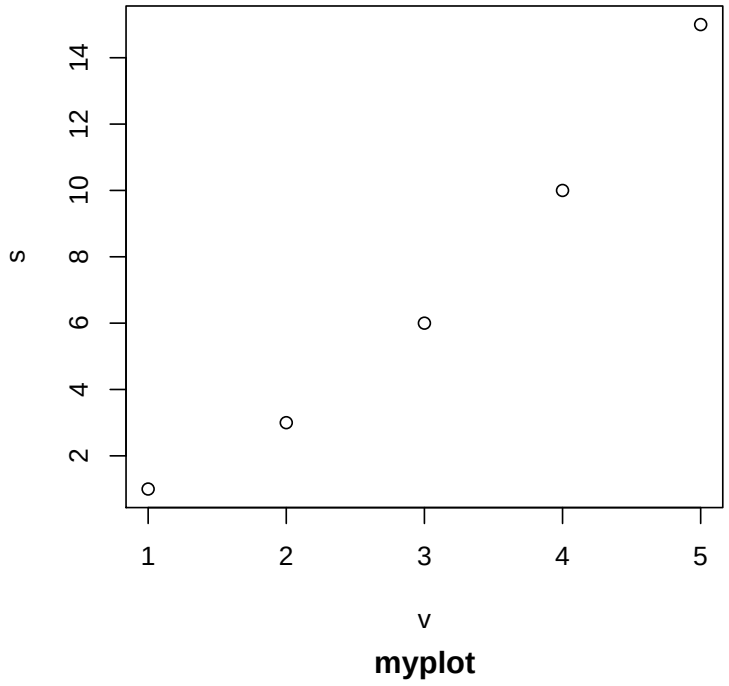
```
> plot(v,s,main="myplot",
+ ylab="sum",pch="+",col="red")
```





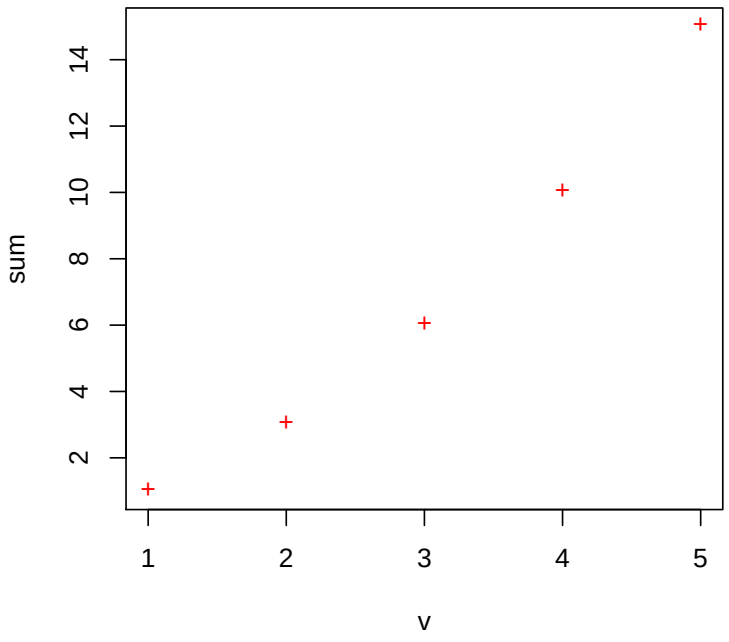
Plotting with R

```
> v = c(1,2,3,4,5)
> s = sapply(1:5,function(i)
+ sum(v[1:i]))
> plot(v,s)
```



There are many parameters to modify the plot, e.g.

```
> plot(v,s,main="myplot",
+ ylab="sum",pch="+",col="red")
```



There are different graphical devices, e.g.

`x11()` (screen, default)

`pdf()`

`dev.off()` closes a device

Statistics with R

There are many many statistical tests, distributions, ... implemented in R.

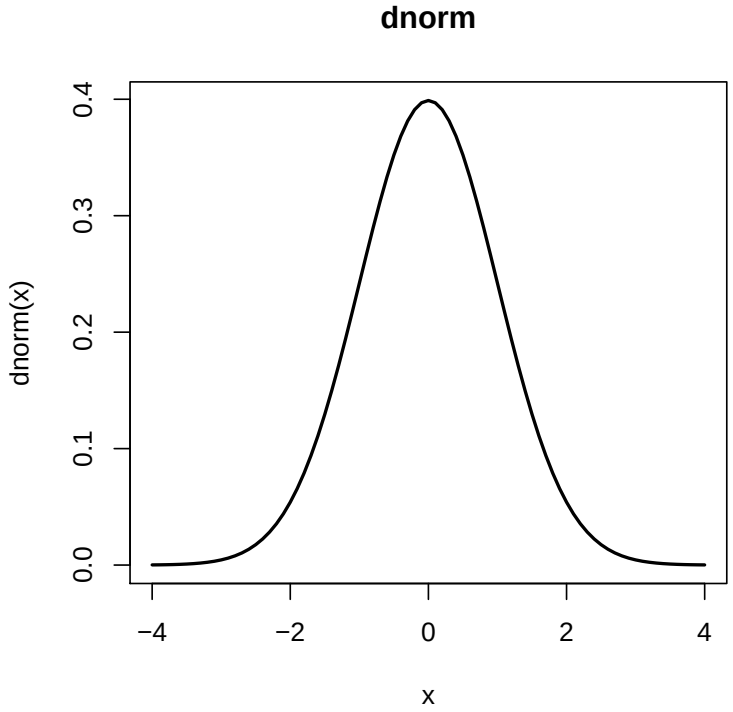
e.g. Fisher-Test on count data:

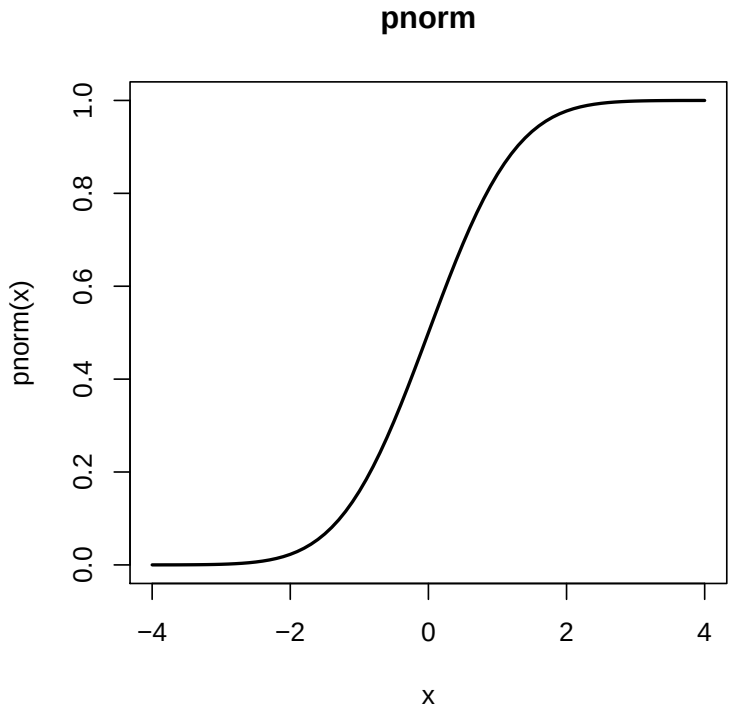
```
> l=matrix(c(2,11,15,5),2)
> l
     [,1] [,2]
[1,]    2   15
[2,]   11    5
> fisher.test(l)

        Fisher's Exact Test for Count Data

data:  l
p-value = 0.001268
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.00547646 0.45530971
sample estimates:
odds ratio
0.06755424
```

Probability and distributions

Distributions can be obtained by:

- probability density d
- cummulative distribution p
- quantiles q
- random r



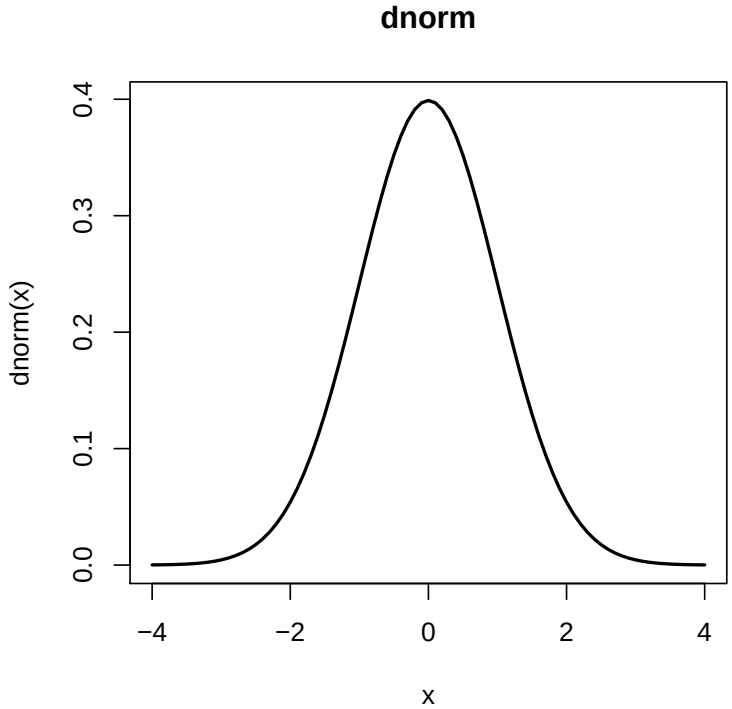


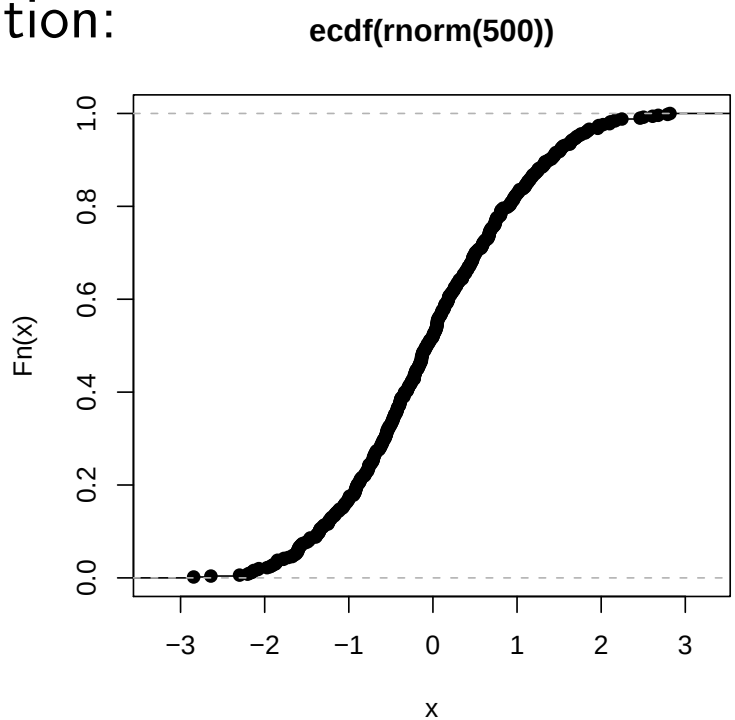
Probability and distributions

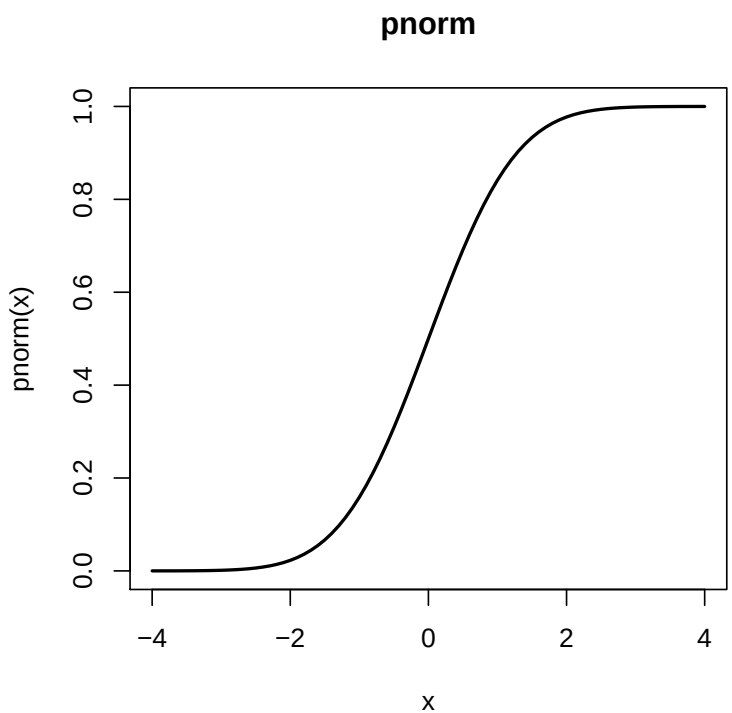
Distributions can be obtained by:

- probability density d
- cummulative distribution p
- quantiles q
- random r

Empirical cummulative distribution:







Literature:

Main source of the information and for further reading:

There are also numerous free documentations of R.
In R, you can get help for a function e.g. `?sum`