EDRP Lab 0

Version: 2020-10-04

Quick introduction to R

This is just a brief introduction to R. There are many great R primers and tutorials on the web. To learn more about R, a good starting place is the homepage of the *R Project for Statistical Computing* at `www.r-project.org`.

What is R?

R is a statistical computing environment for statistical computation and graphics, and it is a computer language designed essentially for statistical and graphical applications. The software is available for unix/linux, Windows, and Macintosh platforms under general public license, and the program is available to download from `www.r-project.org`.

There is also an integrated development environment available for R called *RStudio*. It has an efficient editor with syntax highlighting, and a number of other nice features. You can download it from `www.rstudio.com`.

Orientation

When you bring up R, the first window you see is the R console. You can type directly on the console. Each command you type in R is stored in the history. The up arrow key scrolls back along this history; the down arrow scrolls forward.

A more convenient way is to use the RStudio, where you can edit your commands and submit them for execution using `Ctrl+Enter` combination of keys.

Asking for help

Prefacing any R command with ? will produce a help file:

```
?factorial
```

Vectors

The power of R lies in its ability to work with lists of numbers, called *vectors*.

If m and n are integers, the command `m:n` creates a vector of integers from m to n:

```
1:10
```

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

```
-3:5
```

```
## [1] -3 -2 -1  0  1  2  3  4  5
```

```
5:1
```

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

For more control when generating vectors, you can use the sequence command `seq`. Remember that R is case sensitive. All commands are in lowercase letters.

```
seq(1,10)
```

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

```
seq(1,10,2)
```

```
## [1] 1 3 5 7 9
```

```
seq(10,1,-2)
```

```
## [1] 10  8  6  4  2
```

Create and manipulate vectors using the `c` concatenate command:

```
c(2,1,1,0,9,8)
```

```
## [1] 2 1 1 0 9 8
```

Assign a vector to a variable with the assignment operator `<-`:

```
firstfiveprimes<-c(2,3,5,7,11)
firstfiveprimes
```

```
## [1]  2  3  5  7 11
```

Elements of vectors are indexed with brackets `[]`. Bracket arguments can be single integers or vectors:

```
firstfiveprimes[3]
```

```
## [1] 5
```

```
firstfiveprimes[1:3]
```

```
## [1] 2 3 5
```

```
firstfiveprimes[c(1,4,5,100)]
```

```
## [1]  2  7 11 NA
```

There is no hundredth element in `firstfiveprimes` so R returns `NA` (not available).

The `which` command returns the indices of a vector that satisfy a given property:

```
which(firstfiveprimes<6)
```

```
## [1] 1 2 3
```

```
ind<-which(firstfiveprimes<6)
ind
```

```
## [1] 1 2 3
```

```
firstfiveprimes[ind] #extract the elements that are smaller than 6
```

```
## [1] 2 3 5
```

The `#` symbol you see above is for comments. Anything appearing after `#` is ignored by R.

Vectors can consist of numbers, characters, strings of characters. . . :

```
y<-c("EDRP", "stands","for","Discrete","Random","Processes")
y
```

```
## [1] "EDRP"      "stands"    "for"       "Discrete"  "Random"    "Processes"
```

```
y[c(4,5,6)]
```

```
## [1] "Discrete"  "Random"    "Processes"
```

When performing mathematical operations on vectors, the entire vector is treated as a single object:

```
a<-seq(0,30,4)
a
```

```
## [1]  0  4  8 12 16 20 24 28
```

```
a+1
```

```
## [1]  1  5  9 13 17 21 25 29
```

```
a*2
```

```
## [1]  0  8 16 24 32 40 48 56
```

```
1/a
```

```
## [1]        Inf 0.25000000 0.12500000 0.08333333 0.06250000 0.05000000 0.04166667
## [8] 0.03571429
```

```
b<-a+(1/a)
b
```

```
## [1]      Inf  4.25000  8.12500 12.08333 16.06250 20.05000 24.04167 28.03571
```

```
a*b
```

```
## [1] NaN  17  65 145 257 401 577 785
```

`Inf` (and `-Inf`) are positive (and negative) infinity. By the way: `NaN` means *Not a Number*. (These apply to numeric values and real and imaginary parts of complex values but not to values of integer vectors.) `Inf` and `NaN` are reserved words in the R language.

Notice that when a single number is added to a vector, the number is added to each element of the vector. When two vectors (of the same length) are added together, then corresponding elements are added. This rule applies to most binary operations.

Many mathematical functions can take vector arguments, returning vectors as output:

```
factorial(1:5)
```

```
## [1]   1   2   6  24 120
```

```
sqrt(seq(0,900,100))
```

```
##  [1]  0.00000 10.00000 14.14214 17.32051 20.00000 22.36068 24.49490 26.45751
##  [9] 28.28427 30.00000
```

Some common (and intuitive) commands for working with vectors:

```
x<-c(12.2,14.2,0.555,2,pi,cos(pi),exp(1),abs(-6))
x
```

```
## [1] 12.200000 14.200000  0.555000  2.000000  3.141593 -1.000000  2.718282
## [8]  6.000000
```

```
sum(x)
```

```
## [1] 39.81487
```

```
mean(x)
```

```
## [1] 4.976859
```

```
length(x)
```

```
## [1] 8
```

```
sort(x)
```

```
## [1] -1.000000  0.555000  2.000000  2.718282  3.141593  6.000000 12.200000
## [8] 14.200000
```

```
sort(x,decreasing=T)
```

```
## [1] 14.200000 12.200000  6.000000  3.141593  2.718282  2.000000  0.555000
## [8] -1.000000
```

Randomness

Random sampling

The `sample` command generates a random sample of a given size from the elements of a vector. The syntax is `sample(vec,size,replace=,prob=)`. Samples can be taken with or without replacement (the default is without). A probability distribution on the elements of `vec` can be specified (the default is that all elements of `vec` are equally likely).

```
sample(1:10,1)
```

```
## [1] 6
```

```
sample(1:10,5)
```

```
## [1] 2 6 9 3 8
```

```
sample(c(-1,0,1),6,replace=T)
```

```
## [1]  0  0  0 -1  1 -1
```

```
sample(1:6,10,replace=T) #a simulation of 10 rolls of an ordinary die
```

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

```
s<-sample(c("A","B","C","D"),30,replace=T,prob=c(0.1,0.2,0.2,0.5)) #what does this mean?
s
```

```
##  [1] "C" "C" "C" "C" "B" "B" "D" "D" "D" "C" "B" "C" "D" "D" "B" "D" "D" "A" "B"
## [20] "D" "A" "C" "A" "B" "B" "D" "D" "D" "D" "B"
```

```
index<-which(s=="D")
index
```

```
##  [1]  7  8  9 13 14 16 17 20 26 27 28 29
```

```
length(index)
```

```
## [1] 12
```

We have used the logical connective `==` - *equal to*. Some other logical connectives:

- `!=` - not equal to
- `>=` - greater than or equal to
- `&` - and
- `|` - or
- `|!|`- not.

Probability distributions

Essentially, there are four commands for working with probability distributions in R. They take the root name of the distribution (see below) and prefix the root with `d`, `p`, `q` or `r`. Meanings of the prefixes:

- `d` - density or probability mass functions
- `p` - (cumulative) distribution function,
- `q` - quantile,
- `r` - random variable.

Some of the roots:

- `norm` - normal,
- `unif` - uniform,
- `binom` - binomial,
- `cauchy` - Cauchy,
- `exp` - exponential,
- `geom` - geometric,
- `pois` - Poisson.

Examples:

```
runif(6,0,1) # generates six random numbers uniformly distributed on (0,1)
```

```
## [1] 0.5568792 0.5010678 0.7364431 0.4647172 0.5227110 0.6418666
```

```
pexp(3,1) # P(X<=3) if X has an exponential distribution with parameter lambda=1
```

```
## [1] 0.9502129
```

```
dbinom(10,20,0.5) # P(X=10) where X has a binomial distribution with parameters n=20 and p=0.5
```

```
## [1] 0.1761971
```

```
qnorm(0.95,0,1)
```

```
## [1] 1.644854
```

Important: convention regarding the normal distribution. In R, you specify the standard deviation, *not* the variance:

```
ns<-rnorm(18,-1,5) # 18 random numbers from the normal distr. with mean -1 and std. dev. 5
ns
```

```
##  [1]   4.2252686  -2.1271823   2.3868312   0.2270908   0.6068022   0.7573587
##  [7]  -5.1106789   1.2349342   2.9807388   4.0310321 -11.0346819  -8.6824264
## [13]  10.6723868  -3.8246667  -1.3567011  -2.8552968  -1.1771493   4.0126191
```

Consistent with:

```
sd(ns) # sample standard deviation
```

```
## [1] 5.050308
```

```r
sd(ns)^2 # the square of the sample standard deviation
```

```
## [1] 25.50561
```

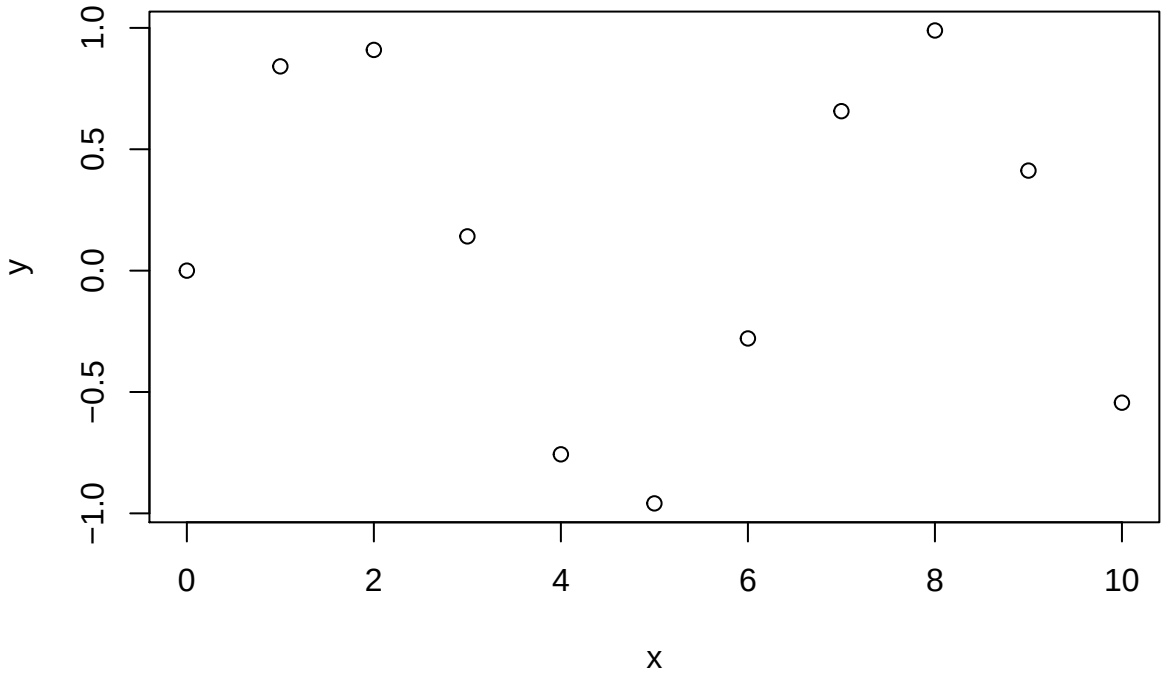
```r
var(ns) # sample variance
```

```
## [1] 25.50561
```

A little bit of graphics

One of the basic commands is `plot`:

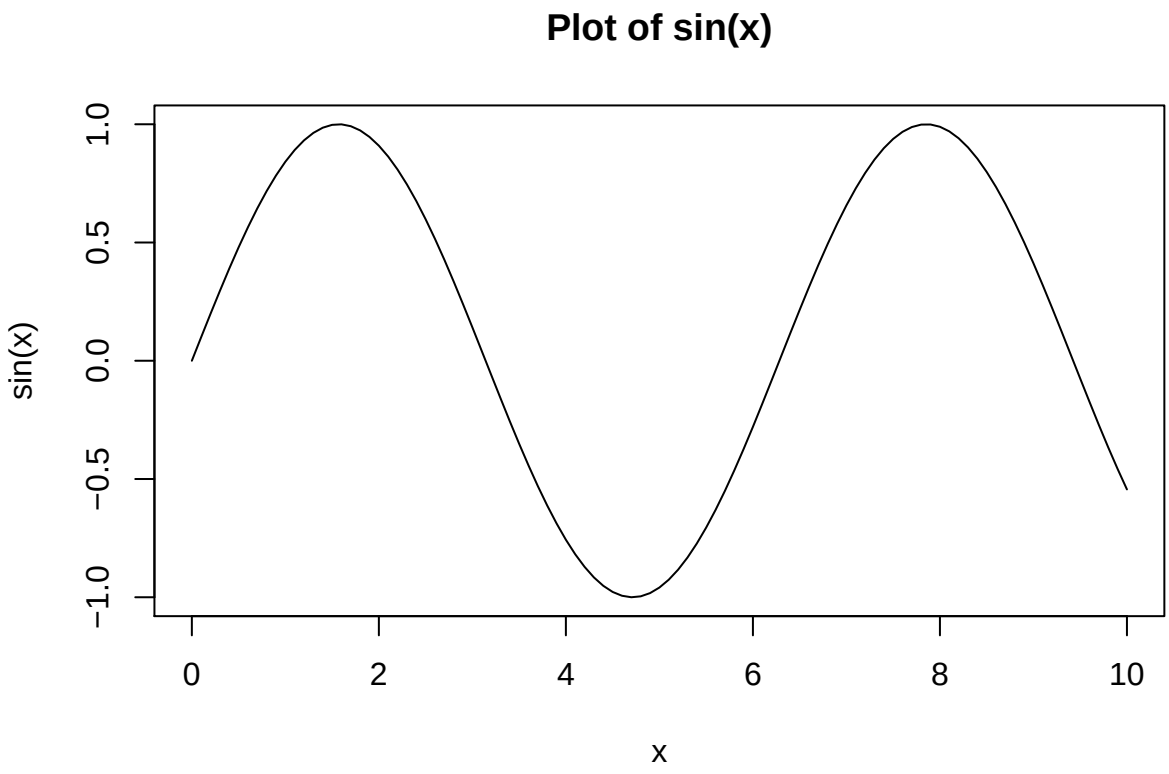
```r
x<-0:10
y<-sin(x)
plot(x,y)
```



Power of `plot` reaches well beyond the above simple example.

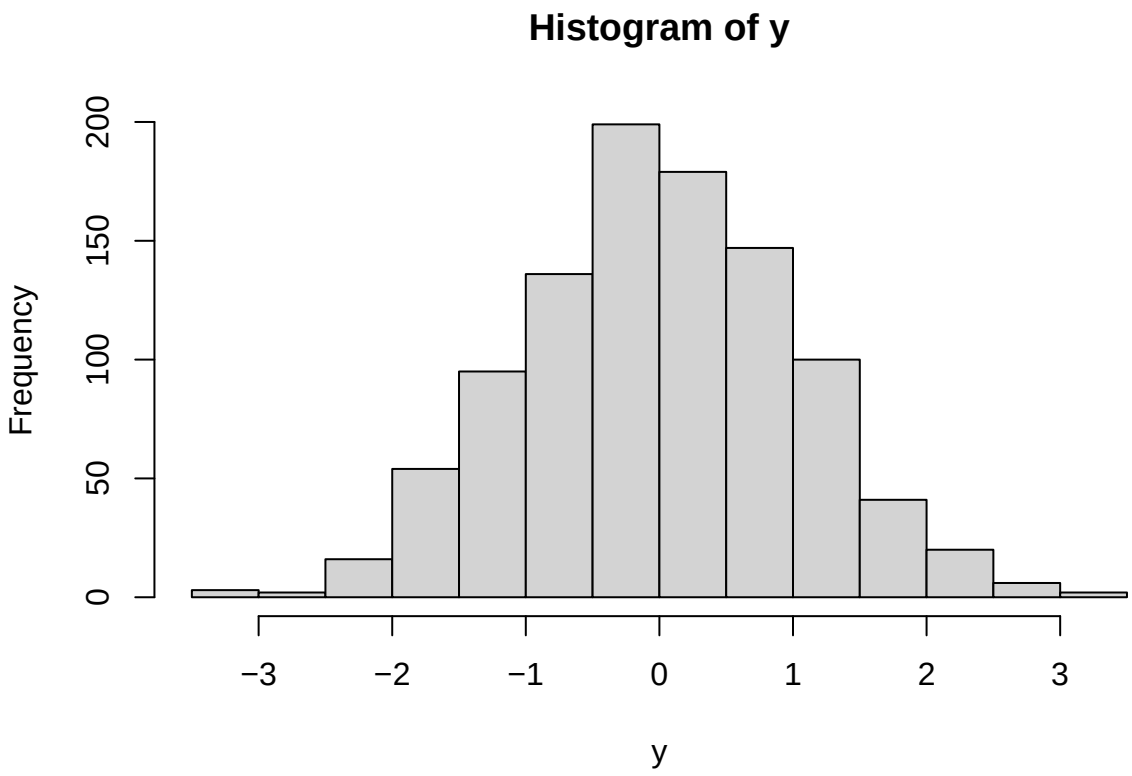
To graph smooth functions, one can use `curve`:

```r
curve(sin(x),0,10,xlab = "x",ylab = "sin(x)",main="Plot of sin(x)")
```

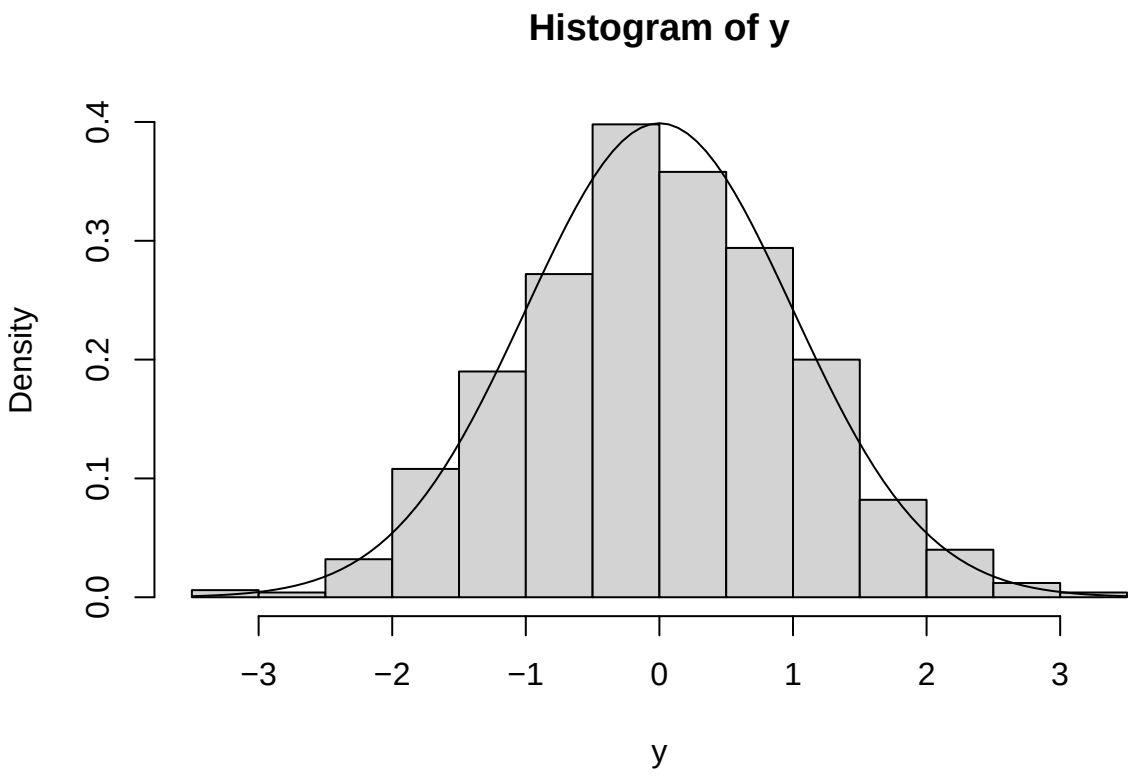


For histograms, one can use `hist(vec)`:

```
y<-rnorm(1000,0,1)
hist(y)
```




```
hist(y,freq=F)
curve(dnorm(x,mean=0,sd=1),add=T)
```



We simulate 1000 observations from the standard normal distribution, plot the histogram, and add the standard normal density curve.

Functions

You can create your own functions in R. The syntax is:

```
name<-function(arg1,arg2,...) expressions
```

Some examples:

```
area<-function(radius) pi*radius^2
area(1)
```

```
## [1] 3.141593
```

```
easyf<-function(x,y,z) x+y+z
easyf(1,2,3)
```

```
## [1] 6
```

If the function definition contains more than one line of code, enclose the code in curly braces `{}`:

```
threesums<-function(vec)
{
  s1<-sum(vec)
  s2<-sum(vec^2)
  s3<-sum(vec^3)
  c(s1,s2,s3)
```

```
}
threesums(1:5)
```

```
## [1]  15  55 225
```

Some other useful commands

```
table(c(1,1,1,2,2,3,3,3,3)) #creates a frequency table
```

```
##
## 1 2 3
## 3 2 4
```

```
table(1:5)
```

```
##
## 1 2 3 4 5
## 1 1 1 1 1
```

```
min(0:20)
```

```
## [1] 0
```

```
max(0:20)
```

```
## [1] 20
```

```
round(pi,5)
```

```
## [1] 3.14159
```

```
replicate(3,0.1)
```

```
## [1] 0.1 0.1 0.1
```

```
# syntax: replicate(n,expr) - the expression expr is repeated n times and output as a vector
results <-replicate(50,{
sample<-rnorm(10) #a random sample from N(0,1) of size 10
sd(sample)
})
results
```

```
##  [1] 0.8672475 1.3610425 0.6867850 1.2463138 0.9103635 0.9858938 0.7817600
##  [8] 0.8836367 0.9375467 1.3317026 0.9954294 0.7532902 1.1053511 1.1449029
## [15] 1.0259380 0.7774709 1.3550816 0.8964622 0.9255725 1.0893689 0.6072339
## [22] 1.0319856 1.0176689 0.8904053 1.2758708 1.0252435 0.9703252 1.2908070
## [29] 0.7675367 0.6238128 0.8675887 0.8228428 1.2121057 0.7959205 0.7446701
## [36] 0.9030266 1.2026170 0.9227936 0.7233955 1.0219869 0.9184641 1.3472333
## [43] 1.1463823 0.7127844 0.7630497 0.8177684 0.4857925 0.7641826 0.6209383
## [50] 1.1609498
```

A Poisson distribution with parameter λ has mean $\mu = \lambda$ and standard deviation $\sigma = \sqrt{\lambda}$. We generate $n = 80$ observations $X_1, \ldots, X_{80}$ from a Poisson distribution with parameter $\lambda = 4$. To illustrate the Central Limit Theorem we compute the normalized sum

$$\frac{(X_1 + \ldots + X_n) - n\mu}{\sigma\sqrt{n}}.$$

```
normalized_sum<-function()(
  (mean(rpois(80,4))-4)/(2/sqrt(80))
)
normalized_sum()
```

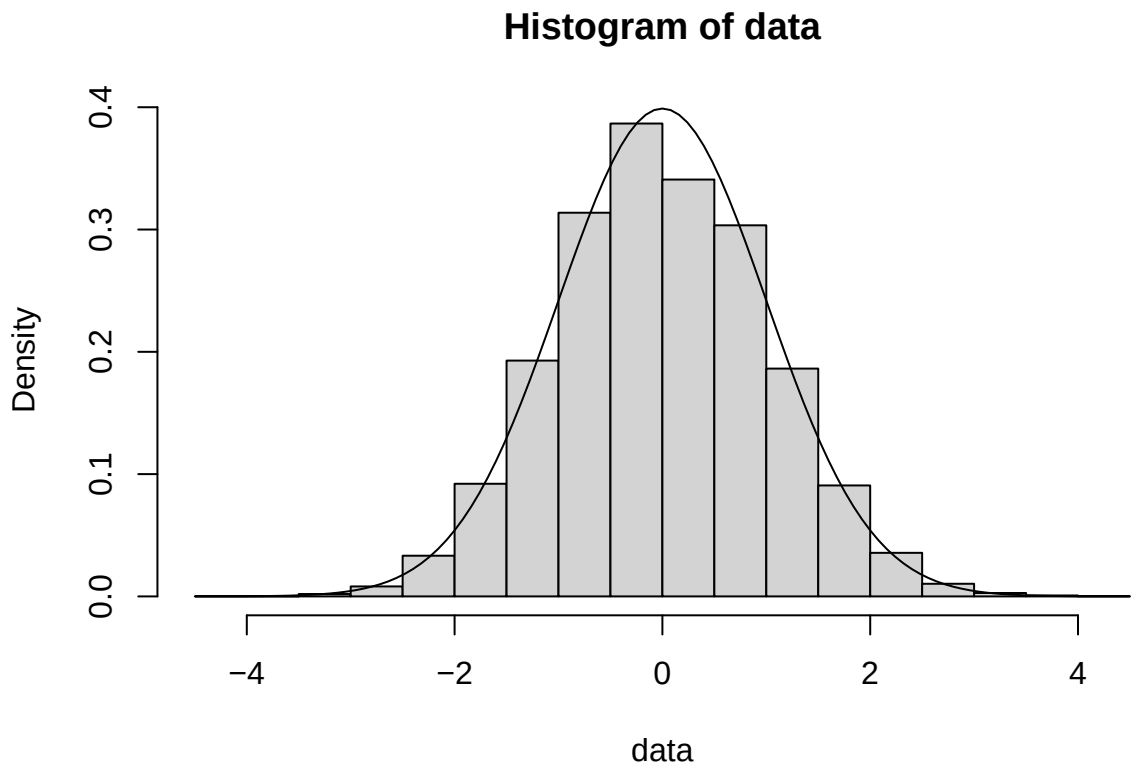
```
## [1] 0.4472136
```

```
normalized_sum()
```

```
## [1] 1.565248
```

We repeat the simulation 100000 times and graph the resulting data as a histogram together with the density of the standard normal distribution:

```
data<-replicate(100000,normalized_sum())
hist(data,freq=F)
curve(dnorm(x),-4,4,add=T)
```



Working with matrices

The `matrix` command is used to create matrices from vectors. The default is to fill the matrix by columns.

```
matrix(1:9,nrow=3)
```

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

```
matrix(1:9,nrow=3,byrow=T)
```

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

Operations on matrices:

```
A<-matrix(c(1,3,0,-4),ncol = 2,byrow = T)
B<-matrix(1:4,ncol = 2, byrow = T)
A
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    0   -4
```

```
B
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

```
2+A
```

```
##      [,1] [,2]
## [1,]    3    5
## [2,]    2   -2
```

```
2*B
```

```
##      [,1] [,2]
## [1,]    2    4
## [2,]    6    8
```

```
A+B
```

```
##      [,1] [,2]
## [1,]    2    5
## [2,]    3    0
```

```
A-B
```

```
##      [,1] [,2]
## [1,]    0    1
## [2,]   -3   -8
```

```
A*B
```

```
##      [,1] [,2]
## [1,]    1    6
## [2,]    0  -16
```

```
A/B
```

```
##      [,1] [,2]
## [1,]    1  1.5
## [2,]    0 -1.0
```

```
2^B
```

```
##      [,1] [,2]
## [1,]    2    4
```

```
## [2,]    8   16
```

The matrix multiplication operator `%*%`:

```
A %*% A
```

```
##      [,1] [,2]
## [1,]    1   -9
## [2,]    0   16
```

```
x<-c(2,1)
A %*% x
```

```
##      [,1]
## [1,]    5
## [2,]   -4
```

```
y<-matrix(c(1,3),nrow=2)
y %*% x
```

```
##      [,1] [,2]
## [1,]    2    1
## [2,]    6    3
```

The transpose of a matrix:

```
A<-matrix(c(3,1,-2,0,0,4),nrow=2,byrow=T)
A
```

```
##      [,1] [,2] [,3]
## [1,]    3    1   -2
## [2,]    0    0    4
```

```
t(A)
```

```
##      [,1] [,2]
## [1,]    3    0
## [2,]    1    0
## [3,]   -2    4
```

```
A %*% t(A)
```

```
##      [,1] [,2]
## [1,]   14   -8
## [2,]   -8   16
```

To solve the linear system $Ax = b$, type `solve(A,b)`. If A is invertible, `solve(A)` returns the inverse A^{-1}:

```
A<-matrix(c(2,0,1,1,-1,4,3,1,0),nrow=3,byrow=T)
A
```

```
##      [,1] [,2] [,3]
## [1,]    2    0    1
## [2,]    1   -1    4
## [3,]    3    1    0
```

```
b<-c(0,1,1)
solve(A,b)
```

```
## [1] -0.5  2.5  1.0
```

```
solve(A)
```

```
##      [,1]  [,2]  [,3]
## [1,]    1 -0.25 -0.25
## [2,]   -3  0.75  1.75
## [3,]   -1  0.50  0.50
```

```
A %*% solve(A)
```

```
##      [,1] [,2]          [,3]
## [1,]    1    0 -1.110223e-16
## [2,]    0    1 -4.440892e-16
## [3,]    0    0  1.000000e+00
```

Diagonal matrices:

```
diag(3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

```
A<-diag(c(1,2,3))
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    2    0
## [3,]    0    0    3
```

```
diag(A)
```

```
## [1] 1 2 3
```

For an $n \times n$ matrix A, `eigen(A)` returns the eigenvalues and eigenvectors of A in a two-component list. The first component of the list `eigen(A)$values` is a vector containing the n eigenvalues. The second, `eigen(A)$vectors` is a $n \times n$ matrix whose columns contain the corresponding eigenvectors.

Below, we diagonalize the matrix A. That is, we find an invertible matrix S and a diagonal matrix D such that $A = SDS^{-1}$. The diagonal elements of D are the eigenvalues of A. The columns of S are the corresponding eigenvectors.

```
A<-matrix(replicate(9,1),nrow = 3)
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    1    1
## [2,]    1    1    1
## [3,]    1    1    1
```

```
eigen(A)$values
```

```
## [1] 3.000000e+00 8.881784e-16 0.000000e+00
```

```
eigen(A)$vectors
```

```
##            [,1]       [,2]       [,3]
## [1,] -0.5773503  0.8164966  0.0000000
## [2,] -0.5773503 -0.4082483 -0.7071068
## [3,] -0.5773503 -0.4082483  0.7071068
```

```
D<-diag(eigen(A)$values)
D
```

```
##      [,1]         [,2] [,3]
## [1,]    3 0.000000e+00    0
## [2,]    0 8.881784e-16    0
## [3,]    0 0.000000e+00    0
```

```
S<-eigen(A)$vectors
S
```

```
##            [,1]       [,2]       [,3]
## [1,] -0.5773503  0.8164966  0.0000000
## [2,] -0.5773503 -0.4082483 -0.7071068
## [3,] -0.5773503 -0.4082483  0.7071068
```

```
S %*% D %*% solve(S)
```

```
##      [,1] [,2] [,3]
## [1,]    1    1    1
## [2,]    1    1    1
## [3,]    1    1    1
```

Data frames

Data frames are used to store tabular data in R. They are an important type of object in R.

Data frames are represented as a special type of list where every element of the list has to have the same length. Each element of the list can be thought of as a column and the length of each element of the list is the number of rows.

Unlike matrices, data frames can store different classes of objects in each column. Matrices must have every element be the same class (e.g. all integers or all numeric).

```
df<-data.frame(a=1:4,b=c(T,F,F,T),x5=c(0.4,0.2,0.5,0.7))
df
```

```
##   a     b  x5
## 1 1  TRUE 0.4
## 2 2 FALSE 0.2
## 3 3 FALSE 0.5
## 4 4  TRUE 0.7
```

```
df$a
```

```
## [1] 1 2 3 4
```

```
df$b
```

```
## [1]  TRUE FALSE FALSE  TRUE
```

```
colnames(df)
```

```
## [1] "a"  "b"  "x5"
```

An example: we create a long list of random numbers from the uniform distribution on $[0, 1]$ by concatenating 10 shorter lists (with 5 elements each).

```
A<-runif(5,0,1)
for (i in 2:10)
```

```
{
B<-runif(5,0,1)
A<-c(A,B)
}
A
```

```
##  [1] 0.10239621 0.77559262 0.27113870 0.07384924 0.67533524 0.24527695
##  [7] 0.53638613 0.11209529 0.24948858 0.43800027 0.27416694 0.71217229
## [13] 0.74120175 0.80297498 0.37142161 0.96465531 0.42499426 0.59238984
## [19] 0.97583448 0.51294930 0.02015275 0.83176770 0.83095156 0.95950153
## [25] 0.91624012 0.17397745 0.48494705 0.83871926 0.47762869 0.57370699
## [31] 0.01965254 0.81261148 0.43890620 0.86150991 0.67401530 0.88220389
## [37] 0.40218325 0.18042066 0.42629507 0.39079684 0.79514335 0.60328788
## [43] 0.13304403 0.04243228 0.93952779 0.37329121 0.48491587 0.26928443
## [49] 0.12201556 0.71776971
```

Next we create a data frame from the long list, with 10 rows (so every short list becomes a row in the data frame):

```
df <- data.frame(matrix(unlist(A), nrow=10, byrow=T))
df
```

```
##            X1        X2        X3         X4        X5
## 1  0.10239621 0.7755926 0.2711387 0.07384924 0.6753352
## 2  0.24527695 0.5363861 0.1120953 0.24948858 0.4380003
## 3  0.27416694 0.7121723 0.7412017 0.80297498 0.3714216
## 4  0.96465531 0.4249943 0.5923898 0.97583448 0.5129493
## 5  0.02015275 0.8317677 0.8309516 0.95950153 0.9162401
## 6  0.17397745 0.4849471 0.8387193 0.47762869 0.5737070
## 7  0.01965254 0.8126115 0.4389062 0.86150991 0.6740153
## 8  0.88220389 0.4021832 0.1804207 0.42629507 0.3907968
## 9  0.79514335 0.6032879 0.1330440 0.04243228 0.9395278
## 10 0.37329121 0.4849159 0.2692844 0.12201556 0.7177697
```

Now we are able to easily find the number of rows in which, for example, $X1 > 0.5$ and $X3 < 0.5$:

```
sum(df$X1 > 0.5 & df$X3 < 0.5) # upper case letters for the variable names
```

```
## [1] 2
```

If we want to rename the columns, we can write:

```
names(df)<-c("x0", "a", "bcd", "x3", "Ux")
head(df)
```

```
##           x0         a       bcd         x3        Ux
## 1 0.10239621 0.7755926 0.2711387 0.07384924 0.6753352
## 2 0.24527695 0.5363861 0.1120953 0.24948858 0.4380003
## 3 0.27416694 0.7121723 0.7412017 0.80297498 0.3714216
## 4 0.96465531 0.4249943 0.5923898 0.97583448 0.5129493
## 5 0.02015275 0.8317677 0.8309516 0.95950153 0.9162401
## 6 0.17397745 0.4849471 0.8387193 0.47762869 0.5737070
```