APA-Intro-a-R

September 6, 2018

1 APA Session 0 - Introduction to R

```
In [1]: # default figure size
        options(repr.plot.width=6, repr.plot.height=6)
```

1.1 R Getting help

One of the first things you would like to do is getting help

Given an R command or function x, entering ?x will bring up its help file

```
In [2]: ?exp
```

The next command searches local matches for the string 'exponential' and provides a list of help files in which this term can be found in the concept or title (note that in R single '' and double "" quotes are the same)

```
In [3]: help.search("exponential")

        help.start()        # hypertext (HTML) version of R's online doc

        RSiteSearch("mean") # broad search for 'mean' on http://search.r-project.org (will open

        help.search("mean")
        ??mean              # `??' is a shortcut for `help.search()'
```

```
starting httpd help server ... done


If the browser launched by '/usr/bin/xdg-open' is already running, it
    is *not* restarted, and you must switch to its window.
Otherwise, be patient ...
A search query has been submitted to http://search.r-project.org
The results page should open in your browser shortly
```

```
In [4]: help(mean)      # help on function mean in local search path
        ?mean           # shortcut for `help(mean)'
```

```
In [5]: example(mean)  # run all the examples proposed in `help(mean)'
```

```
mean> x <- c(0:10, 50)

mean> xm <- mean(x)

mean> c(xm, mean(x, trim = 0.10))
[1] 8.75 5.50
```

```
In [6]: apropos("str") # everythng containing 'str' in its name
```

1. 'austres' 2. 'constrOptim' 3. 'default.stringsAsFactors' 4. 'encodeString' 5. 'isTRUE' 6. 'ls.str' 7. 'lsf.str' 8. 'NLSstRtAsymptote' 9. 'R.version.string' 10. 'str' 11. 'strcapture' 12. 'stream_in' 13. 'stream_out' 14. 'strftime' 15. 'strheight' 16. 'stripchart' 17. 'strOptions' 18. 'strptime' 19. 'strrep' 20. 'strsplit' 21. 'strtoi' 22. 'strtrim' 23. 'StructTS' 24. 'structure' 25. 'strwidth' 26. 'strwrap' 27. 'substr' 28. 'substr<-' 29. 'substring' 30. 'substring<-' 31. 'toString' 32. 'toString.default'

1.2 Expresions evaluation

```
In [7]: 3-    1 # R will ignore these spaces

        3 -     # the '+' prompt signals an incomplete command
        1
```

2
2

R handles special situations explicitly

```
In [8]: -1/0; 0/0; 1/0; sqrt(-2+0i) # use ';' to evaluate distinct expressions

        1:5          # The numbers 1, 2, 3, 4, 5

        mean(1:5)

        sum(1:5)     # Apply the sum function to the vector of numbers 1, 2, 3, 4, 5

        (2:5)^10     # 2 to the power of 10, 3 to the power of 10, ...

        log2(c(0.5, 1, 2, 4, 8))
```

-Inf
NaN
Inf
0+1.4142135623731i

1. 1 2. 2 3. 3 4. 4 5. 5
3
15
1. 1024 2. 59049 3. 1048576 4. 9765625
1. -1 2. 0 3. 1 4. 2 5. 3

R is **case-sensitive**, so foo, Foo and FOO are all considered different names

```
In [9]: x <- 3

        x
```

3

The following code results in an error

```
In [10]: X
```

```
    Error in eval(expr, envir, enclos): objeto 'X' no encontrado
Traceback:
```

1.3 Irregular/regular sequences

```
In [11]: a <- c(1,3,2,4,5)  # function concatenate 'c()'
         a
```

1. 1 2. 3 3. 2 4. 4 5. 5

the () forces to print the evaluation result

```
In [12]: (a <- c(1,3,2,4,5))
```

1. 1 2. 3 3. 2 4. 4 5. 5

```
In [13]: c(0,2^2,mean(a),pi,"hello")
```

1. '0' 2. '4' 3. '3' 4. '3.14159265358979' 5. 'hello'

```
In [14]: 1:5                # forward sequence with operator ':'
         5:1                # backward sequence with operator ':'
         2*5:1              # operator ':' dominates other operators
```

1. 1 2. 2 3. 3 4. 4 5. 5
1. 5 2. 4 3. 3 4. 2 5. 1
1. 10 2. 8 3. 6 4. 4 5. 2

```
In [15]: seq(from=0,to=10,length=5)   # function 'seq()' generates
         seq(from=0,to=10,by=5)       # sorted sequences of given length/step
```

1. 0 2. 2.5 3. 5 4. 7.5 5. 10
1. 0 2. 5 3. 10

1.4 Repeated sequences

```
In [16]: rep(x=1:3,times=2)      # function 'rep()' generates repeated sequences
         rep(x=1:3,times=1:3)
         rep(c("ETH","ZURICH"),c(3,2))
```

1. 1 2. 2 3. 3 4. 1 5. 2 6. 3
1. 1 2. 2 3. 2 4. 3 5. 3 6. 3
1. 'ETH' 2. 'ETH' 3. 'ETH' 4. 'ZURICH' 5. 'ZURICH'

```
In [17]: paste("X",1:5,sep="")              # function 'paste()' and recycling of values
         paste(c("Jan","Feb"),2010,sep="-")
```

1. 'X1' 2. 'X2' 3. 'X3' 4. 'X4' 5. 'X5'
1. 'Jan-2010' 2. 'Feb-2010'

```
In [18]: gl(n=3,k=2,label=c("L","M","H"))    # function 'gl()' generates levels
                                             # (see 'factor()' below)
```

1. L 2. L 3. M 4. M 5. H 6. H
Levels: 1. 'L' 2. 'M' 3. 'H'

1.5 Random sequences

```
In [19]: set.seed(1234) # set seed of the random number generator

         rnorm(n=5,mean=0,sd=2) # 5 draws from N(0,2) with 'rnorm()'

         sample(x=1:10,size=5) # 'sample()' draws samples of specified size

         sort(sample(x=1:10,size=5)) # 'sort()' sorts in ascending/descending order
```

1. -2.41413149877084 2. 0.55485848422132 3. 2.16888235336611 4. -4.6913954052587 5. 0.8582493776221
1. 7 2. 5 3. 3 4. 10 5. 2
1. 2 2. 3 3. 7 4. 9 5. 10

1.6 directory/save/delete

```
In [20]: ls()                  # list stored objects in current R session
         ls(pattern="^x")      # list objects starting with 'x'

         getwd()                # display current working directory
```

1. 'a' 2. 'x' 3. 'xm'
1. 'x' 2. 'xm'
'/home/bejar/PycharmProjects/APA-Notebooks/APA R'

```
In [21]: setwd(dir="/home/nbuser/")     # set current working directory at 'C:\'
                              # use '\\' or '/' instead of '\' in paths

         save(a,file="M.Rdata")         # save R object m in file M.Rdata
         save.image("myfile.Rdata")     # save current workspace in myfile.RData
         savehistory("myhist.Rhistory") # save command history in myhist.Rhistory
```

```
    Error in setwd(dir = "/home/nbuser/"): no es posible cambiar el directorio de trabajo
Traceback:


    1. setwd(dir = "/home/nbuser/")
```

```
In [22]: load("myfile.Rdata");ls()      # load myfile.RData's content into workspace
         loadhistory("myhist.Rhistory") # load command history from hist
```

```
Warning message in readChar(con, 5L, useBytes = TRUE):
cannot open compressed file 'myfile.Rdata', probable reason 'No such file or directory'
```

```
    Error in readChar(con, 5L, useBytes = TRUE): no se puede abrir la conexión
Traceback:


    1. load("myfile.Rdata")

    2. readChar(con, 5L, useBytes = TRUE)
```

1. 'a' 2. 'x' 3. 'xm'

```
In [23]: write("print('Hello!')","hello.txt") # write 'print(Hello!)' in file "hello.txt"
         source("hello.txt")  # execute all R commands in file "hello.txt"
```

```
[1] "Hello!"
```

1.7 Vectors

```
In [24]: vector(mode = "logical",  # 'vector()' generates a vector of given mode
         length = 3)               #  and length with default components

         numeric(10);logical(3)   # 'numeric()', logical(), etc...
                                  # generate logical/logical vectors of specified length

         x <- as.vector(c(2,4,8)); x; mode(x)  # 'as.vector()' transforms
                                               # concatenated object in vector
                                               # 'mode()' displays the vector's mode
```

1. FALSE 2. FALSE 3. FALSE
1. 0 2. 0 3. 0 4. 0 5. 0 6. 0 7. 0 8. 0 9. 0 10. 0
1. FALSE 2. FALSE 3. FALSE
1. 2 2. 4 3. 8
'numeric'

```
In [25]: (names(x) <- c("banana", "apple","nut")) # 'names()' sets components' name
         names(x); is.vector(x); length(x)        # 'names()' also displays the
                                                  # name/labels of the vector's elements
                                                  # 'is.vector()' tests vector's identity,
                                                  # 'length()' displays vector's length
         x.old <- x

         names(x) <- NULL          # erase the names of x's components

         length(x) <- 5            # set the length of x to 5

         x
```

1. 'banana' 2. 'apple' 3. 'nut'
1. 'banana' 2. 'apple' 3. 'nut'
TRUE
3
1. 2 2. 4 3. 8 4. <NA> 5. <NA>

```
In [26]: mode(x)                      # NAs do not alter vector's mode

         mode(x) <- "character"    # change numeric vector to
                                      # character (note that NA's are not converted)

         x
```

'numeric'
1. '2' 2. '4' 3. '8' 4. NA 5. NA

```
In [27]: x <- as.numeric(x);length(x) <- 3  # change character vector to
                                             # numeric and shorten vector to length 3

         x
```

1. 2 2. 4 3. 8

```
In [28]: x[2]; x[-2]; x[c(1,3)]; x[rep(1:3,2)]    # index of positive/negative integers

         x[c(TRUE,TRUE,FALSE)]       # index of logical values TRUE/FALSE
         x[(x>=2)&(x<=6)]

         x.old[c("nut","banana")]     # index vector of character strings
```

```
x.old[c("nut","banana")] <- c(9,0); # modifies these components

x.old
```

4
1. 2 2. 8
1. 2 2. 8
1. 2 2. 4 3. 8 4. 2 5. 4 6. 8
1. 2 2. 4
1. 2 2. 4
nut 8 **banana** 2
banana 0 **apple** 4 **nut** 9

1.8 Factors

```
In [29]: gender <- rep(c("F","M"),c(3,2)); gender; mode(gender) # generate a character vector
```

1. 'F' 2. 'F' 3. 'F' 4. 'M' 5. 'M'
'character'

```
In [30]: levels(gender)   # 'levels()' displays factor levels
                          # vector have no levels

         gender <- factor(gender); gender # 'factor()' converts vectors into factors
```

NULL

1. F 2. F 3. F 4. M 5. M
Levels: 1. 'F' 2. 'M'

```
In [31]: as.character(gender)             # back to the character vector

         cont <- cut(1:9,breaks=3); cont # 'cut()' code the components of the
          # sequence according to specified breaks
          # and it creates a factor
```

1. 'F' 2. 'F' 3. 'F' 4. 'M' 5. 'M'
1. (0.992,3.67] 2. (0.992,3.67] 3. (0.992,3.67] 4. (3.67,6.33] 5. (3.67,6.33] 6. (3.67,6.33] 7. (6.33,9.01] 8. (6.33,9.01] 9. (6.33,9.01]
Levels: 1. '(0.992,3.67]' 2. '(3.67,6.33]' 3. '(6.33,9.01]'

```
In [32]: # much better
         (cont <- cut(1:9,breaks=3,labels=c("low","medium","high")))
```

1. low 2. low 3. low 4. medium 5. medium 6. medium 7. high 8. high 9. high
Levels: 1. 'low' 2. 'medium' 3. 'high'

1.9 Matrices

```
In [33]: A <- matrix(1:16,4,4)
         A
         A[2,3]
         A[c(1,3),c(2,4)]
         A[1:3,2:4]
         A[1:2,]
```

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

10

5	13
7	15

5	9	13
6	10	14
7	11	15

1	5	9	13
2	6	10	14

```
In [34]: A[,1:2]
         A[1,]
         A[-c(1,3),]
         A[-c(1,3),-c(1,3,4)]
         dim(A)
```

1	5
2	6
3	7
4	8

1. 1 2. 5 3. 9 4. 13

2	6	10	14
4	8	12	16

1. 6 2. 8

1. 4 2. 4

```
In [35]: (B <- matrix(1:6,nrow=2,ncol=3,byrow=TRUE))
         (B <- matrix(1:6,nrow=2,ncol=3,byrow=FALSE))

         B[-(2:3), ]


         B[-(2:3), 2]
```

1	2	3
4	5	6

1	3	5
2	4	6

1. 1 2. 3 3. 5

3

```
In [36]: rbind(1:3,4:6)        # function 'rbind()' for binding rows

         cbind(1:2,3,4)        # function 'cbind()' with recycling

         array(1:6,dim=c(2,3)) # function 'array()'
```

1	2	3
4	5	6

1	3	4
2	3	4

1	3	5
2	4	6

```
In [37]: V <- 1:6; dim(V) <- c(2,3); V   # function 'dim()' sets/retrieves dimensions

         V2 <- V     # copy 'V' in 'V2'

         dimnames(V)   # function 'dimnames()' sets/rerieves row/col names

         dimnames(V2) <- list(NULL, c("C1","C2","C3")); V2
```

1	3	5
2	4	6

NULL

C1	C2	C3
1	3	5
2	4	6

```
In [38]: V[-2,c(1,3)]; V2[-2,c("C1","C3")]

         X <- matrix(1:4,ncol=2)   # consider X, a squared matrix
         Y <- matrix(4:1,ncol=2)   # consider Y, a squared matrix

         X + Y       # Elementwise addition (The dimensions must agree)
         X * Y       # Elementwise multiplication (The dimensions must agree)
```

1. 1 2. 5

C1 1 **C3** 5

5	5
5	5

4	6
6	4

```
In [39]: X%*%X                     # matrix multiplication operator '%*%'

         apply(X,MARGIN=1,FUN=sum) # function 'apply()' applies function FUN
                                   # to specified MARGIN

         t(X)                      # function 't()' transposes a matrix
```

7	15
10	22

1. 4 2. 6

1	2
3	4

```
In [40]: solve(X) # solve linear system of the form X = b
                  # if b is not specified, compute the inverse of X
```

-2	1.5
1	-0.5

```
In [41]: solve(X, Y) # Solve X B = Y (X is n by k, Y n by m, B k by m)
```

-3.5	-2.5
2.5	1.5

```
In [42]: eigen(X)            # eigenvalues and eigenvectors of X
```

```
eigen() decomposition
$values
[1]  5.3722813 -0.3722813

$vectors
           [,1]       [,2]
[1,] -0.5657675 -0.9093767
[2,] -0.8245648  0.4159736
```

```
In [43]: svd(X)      # Singular value decomposition of X
```

$d 1. 5.46498570421904 2. 0.365966190626257

$u

-0.5760484	-0.8174156
-0.8174156	0.5760484

$v

-0.4045536	0.9145143
-0.9145143	-0.4045536

1.10 Data frames

```
In [44]: # Data frames are matrices, but they allow named columns and non-numerical data

         mystuff <- data.frame(name=c("BoA","BP","CITI","FAN","UBS","AIG"),
                               pro=c(11,10,18,6,5,6),
                               cty=c("US","UK","US","US","CH","US"))
```

```
In [45]: names(mystuff) # names of the variables in the data frame

         dim(mystuff)  # dimension row/col of the data frame

         str(mystuff)  # 'str()' gives the structure of R objects
```

1. 'name' 2. 'pro' 3. 'cty'
1. 6 2. 3

```
'data.frame':        6 obs. of  3 variables:
 $ name: Factor w/ 6 levels "AIG","BoA","BP",..: 2 3 4 5 6 1
 $ pro : num  11 10 18 6 5 6
 $ cty : Factor w/ 3 levels "CH","UK","US": 3 2 3 3 1 3
```

```
In [46]: mystuff[,2]; mystuff[,2][1:3]   # index vectors on the row and col dimensions

         mystuff$pro; mystuff$pro[1:3]   # extract components with operator `$' and names

         mystuff[[2]]; mystuff[[2]][1:3] # operator `[[i]]' extracts the ith component

         mystuff[[2]]                   # '[[.]]' drops names from 2nd component

         mystuff[2]                     # '[.]' keep names of the 2nd component
```

1. 11 2. 10 3. 18 4. 6 5. 5 6. 6
1. 11 2. 10 3. 18
1. 11 2. 10 3. 18 4. 6 5. 5 6. 6
1. 11 2. 10 3. 18
1. 11 2. 10 3. 18 4. 6 5. 5 6. 6
1. 11 2. 10 3. 18
1. 11 2. 10 3. 18 4. 6 5. 5 6. 6

pro
11
10
18
6
5
6

```
In [47]: attach(mystuff); pro; pro[1:3] # component names can now be used directly

         pro/10; mystuff$pro            # changes in attached objects do not modify
                                        # the source object
```

1. 11 2. 10 3. 18 4. 6 5. 5 6. 6
1. 11 2. 10 3. 18
1. 1.1 2. 1 3. 1.8 4. 0.6 5. 0.5 6. 0.6
1. 11 2. 10 3. 18 4. 6 5. 5 6. 6

```
In [48]: detach(mystuff)                # detach dataframe

         mystuff[order(-mystuff$pro),]    # Rows from 'mystuff' displayed in decreasing order wi
```

	name	pro	cty
3	CITI	18	US
1	BoA	11	US
2	BP	10	UK
4	FAN	6	US
6	AIG	6	US
5	UBS	5	CH

Data frames can be subset

```
In [49]: y <- subset(iris, Species == "versicolor", select = Petal.Length)
          summary(y)
```

```
  Petal.Length
 Min.   :3.00
 1st Qu.:4.00
 Median :4.35
 Mean   :4.26
 3rd Qu.:4.60
 Max.   :5.10
```

1.11 Managing missing values

1.11.1 example 1

```
In [50]: Z <- data.frame(x = c(1, 2, 3), y = c(0, 10, NA)); Z

         na.omit(Z)    # function 'na.omit()' removes rows with NAs
                       # see also functions 'na.fail()',
                       # 'na.exclude()', 'na.pass()'.
```

x	y
1	0
2	10
3	NA

x	y
1	0
2	10

```
In [51]: z <- c(1,-99,-999,"."); z
```

1. '1' 2. '-99' 3. '-999' 4. '.'

```
In [52]: z[is.element(z,c(-99,-999,"."))] <- NA; z  # convert '-99' and  '-999'
                                                     # and '.' into NAs
```

1. '1' 2. NA 3. NA 4. NA

```
In [53]: z <- as.numeric(z); z                       # convert 'z' into numeric
```

1. 1 2. <NA> 3. <NA> 4. <NA>

```
In [54]: z[!is.na(z)]                                # z vector without NAs
```

1

1.11.2 example 2

```
In [55]: library(MASS)
         table(is.na(Pima.tr2$bp))
         sapply(Pima.tr2, function(x)sum(is.na(x)))
```

```
FALSE  TRUE
  287    13
```

npreg 0 **glu** 0 **bp** 13 **skin** 98 **bmi** 3 **ped** 0 **age** 0 **type** 0

1.12 Lists

```
In [56]: mylist <- list(Day=c("Mo","Tu"),    # a list of R objects
         Place="Sabadell",eigen(X),obj4=mystuff)
         names(mylist)                 # names of the objects in 'mylist'
```

1. 'Day' 2. 'Place' 3. '' 4. 'obj4'

```
In [57]: mylist$Day                    # display the object Day in 'mylist'
         mylist$Day[2]                 # display the second element from
                                       # 'Day' in 'mylist'
         mylist$obj4$pro               # shows element 'pro' from 'obj4' in 'mylist'
```

1. 'Mo' 2. 'Tu'
'Tu'
1. 11 2. 10 3. 18 4. 6 5. 5 6. 6

```
In [58]: mylist[[3]]$values            # the i-th component can also be accessed
                                       # with the double squared brackets `[[i]]'

         mylist[[1]]; mylist[1]        # again, check the difference
                                       # between `[[.]]' and `[.]'
```

1. 5.37228132326901 2. -0.372281323269014
1. 'Mo' 2. 'Tu'
$Day = 1. 'Mo' 2. 'Tu'

1.13 Control structures

Let's do some iteration and conditioning

```
In [59]: for (i in 1:10) if ((g<-rnorm(1))>0.25) { cat(paste(i, round(g,4),"\n")) } else { cat("
```

```
no!
no!
no!
no!
5 0.9595
no!
no!
no!
no!
10 2.4158
```

1.14 Function definitions

1.14.1 Example 1

```
In [60]: my.sqr <- function(x) x^2 ; my.sqr(1:5)
```

1. 1 2. 4 3. 9 4. 16 5. 25

1.14.2 Example 2

```
In [61]: drawCards = function (number)
         {
           faceValues = c(1:10, "Jack", "Queen", "King")
           suits = c("Hearts", "Diamonds", "Spades", "Clubs")
           f = sample(faceValues, number, replace=TRUE)
           s = sample(suits, number, replace=TRUE)
           paste(f, "of", s)
         }

         drawCards(5)
```

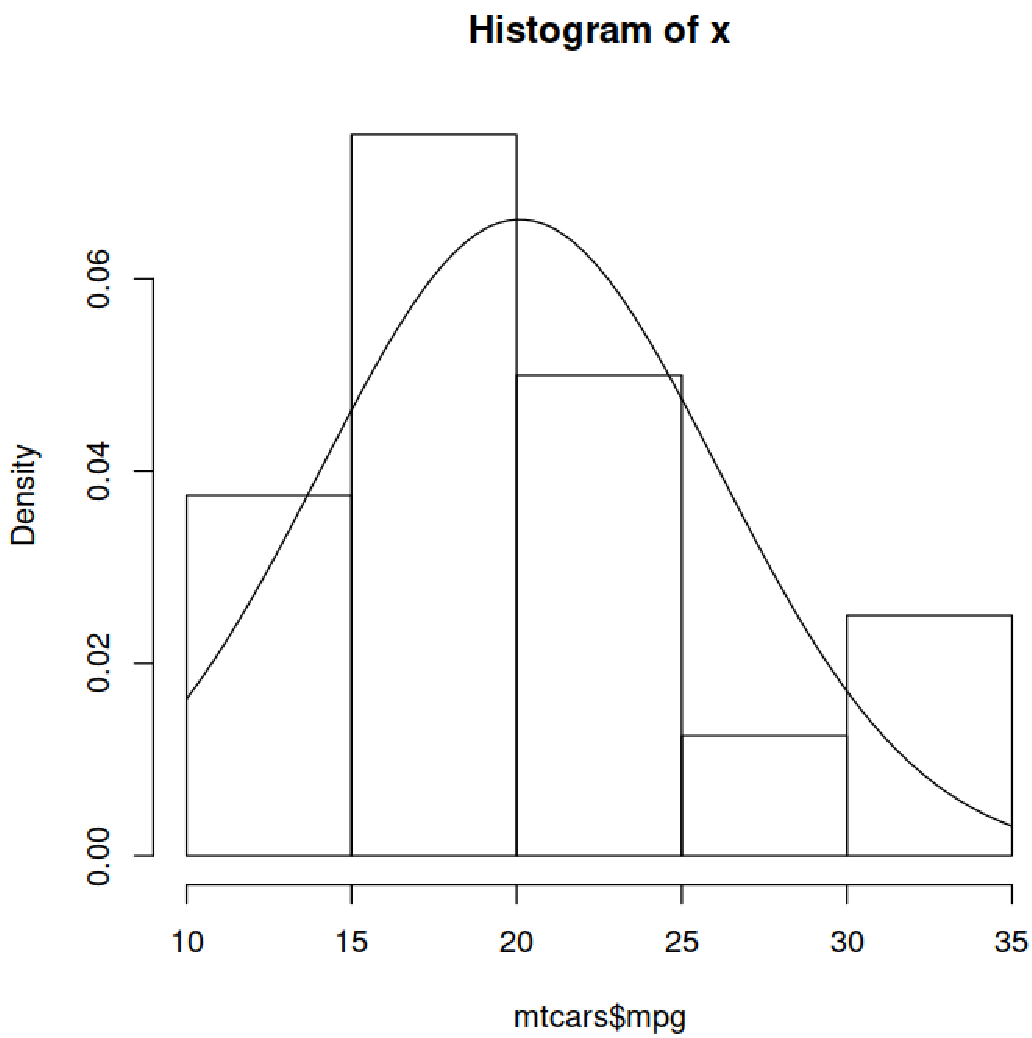
1. '8 of Spades' 2. '9 of Spades' 3. '5 of Diamonds' 4. '9 of Hearts' 5. '5 of Clubs'

Note that the function returns as value the result of its last evaluation

1.14.3 Example 3

Plotting a variable against the normal pdf

In [62]:
```
hist.with.normal <- function (x, xlabel=deparse(substitute(x)), ...)
{
  h <- hist(x,plot=F, ...)
  s <- sd(x)
  m <- mean(x)
  ylim <- range(0,h$density,dnorm(0,sd=s))
  hist(x,freq=F,ylim=ylim,xlab=xlabel, ...)
  curve(dnorm(x,m,s),add=T)
}

hist.with.normal (mtcars$mpg)
```



1.15 Importing/exporting .csv files

```
In [63]: #create a data frame
         dates <- c("3/27/1995", "4/3/1995",
                    "4/10/1995", "4/18/1995")
         prices <- c(11.1, 7.9, 1.9, 7.3)
         d <- data.frame(dates=dates, prices=prices)
```

```
In [64]: #create the .csv file
         dir.create ("labo")
         filename <- "labo/temp.csv"
         write.table(d, file = filename, sep = ",", row.names = FALSE)
```

```
Warning message in dir.create("labo"):
'labo' ya existe
```

```
In [65]: #read the .csv file
         read.table(file = filename, sep = ",", header = TRUE)
         read.csv(file = filename) #same thing
```

dates	prices
3/27/1995	11.1
4/3/1995	7.9
4/10/1995	1.9
4/18/1995	7.3

dates	prices
3/27/1995	11.1
4/3/1995	7.9
4/10/1995	1.9
4/18/1995	7.3

1.16 Processing a data file

Consider the 'trees' data set, that provides measurements of the girth, height and volume of timber in 31 felled black cherry trees.

```
In [66]: trees
```

Girth	Height	Volume
8.3	70	10.3
8.6	65	10.3
8.8	63	10.2
10.5	72	16.4
10.7	81	18.8
10.8	83	19.7
11.0	66	15.6
11.0	75	18.2
11.1	80	22.6
11.2	75	19.9
11.3	79	24.2
11.4	76	21.0
11.4	76	21.4
11.7	69	21.3
12.0	75	19.1
12.9	74	22.2
12.9	85	33.8
13.3	86	27.4
13.7	71	25.7
13.8	64	24.9
14.0	78	34.5
14.2	80	31.7
14.5	74	36.3
16.0	72	38.3
16.3	77	42.6
17.3	81	55.4
17.5	82	55.7
17.9	80	58.3
18.0	80	51.5
18.0	80	51.0
20.6	87	77.0

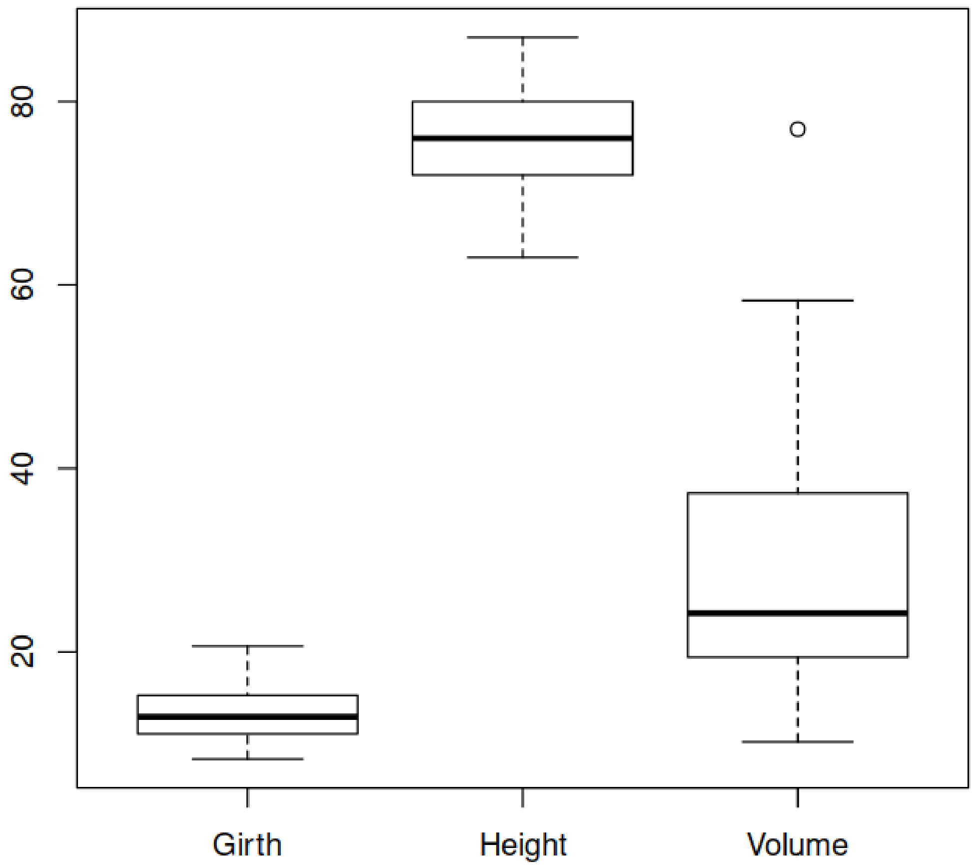
R can provide very quick summary information

```
In [67]: summary(trees)
         names(trees)
```

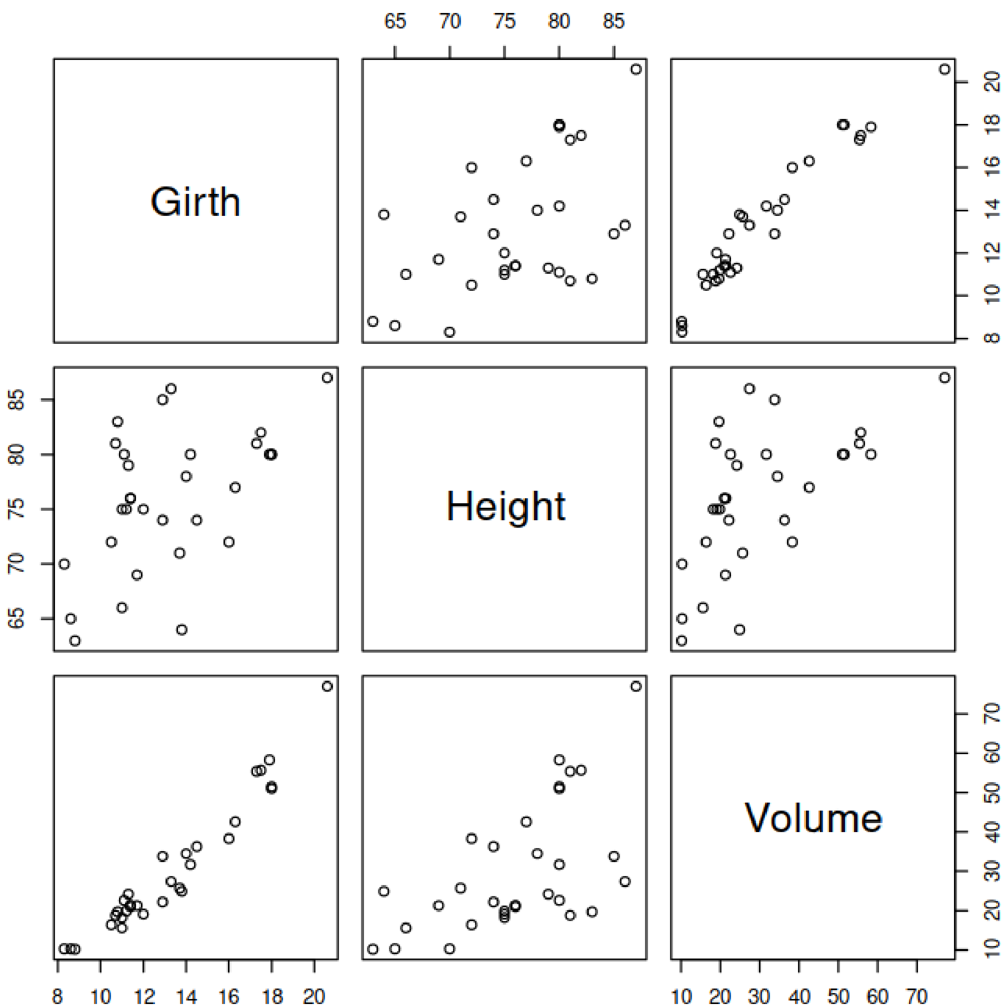
```
     Girth           Height       Volume
 Min.   : 8.30   Min.   :63   Min.   :10.20
 1st Qu.:11.05   1st Qu.:72   1st Qu.:19.40
 Median :12.90   Median :76   Median :24.20
 Mean   :13.25   Mean   :76   Mean   :30.17
 3rd Qu.:15.25   3rd Qu.:80   3rd Qu.:37.30
 Max.   :20.60   Max.   :87   Max.   :77.00
```

1. 'Girth' 2. 'Height' 3. 'Volume'

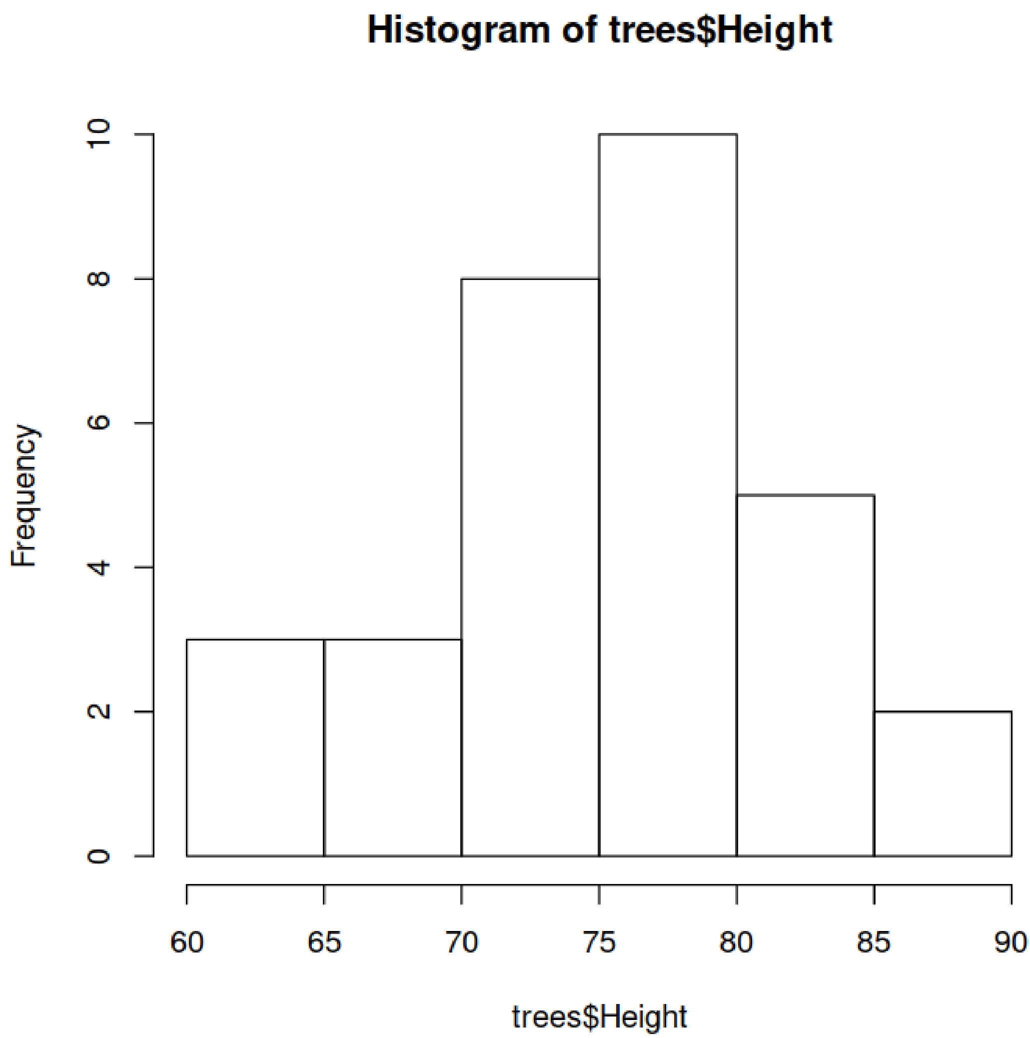
```
In [68]: boxplot(trees)
```



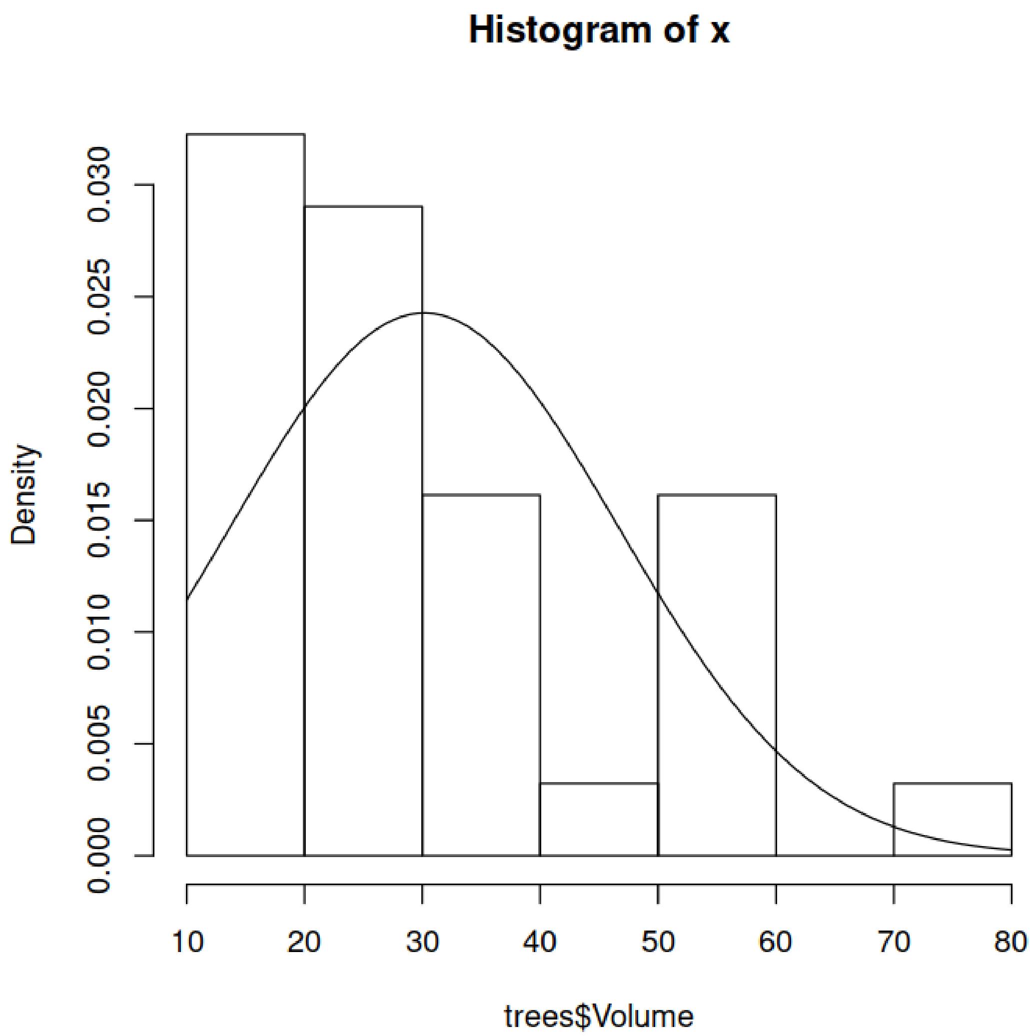

```
In [69]: pairs(trees)
```



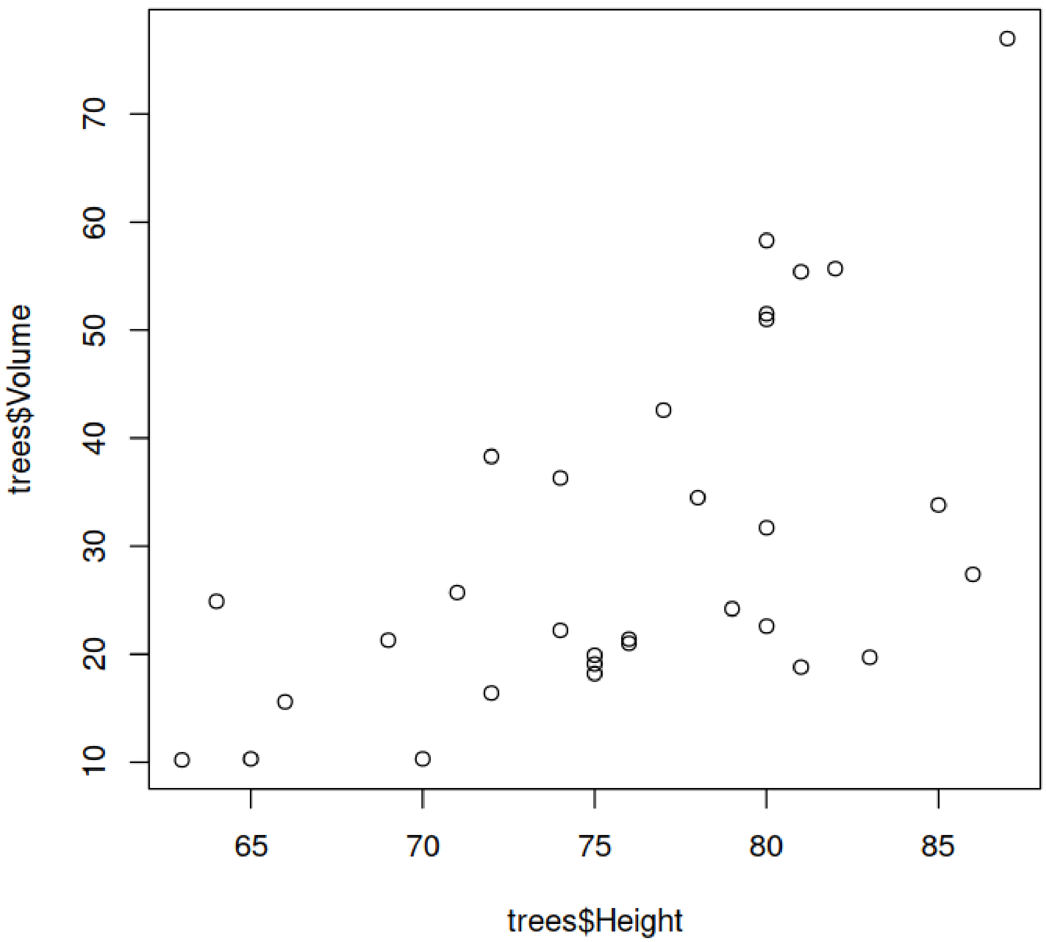

```
In [70]: hist(trees$Height)
```



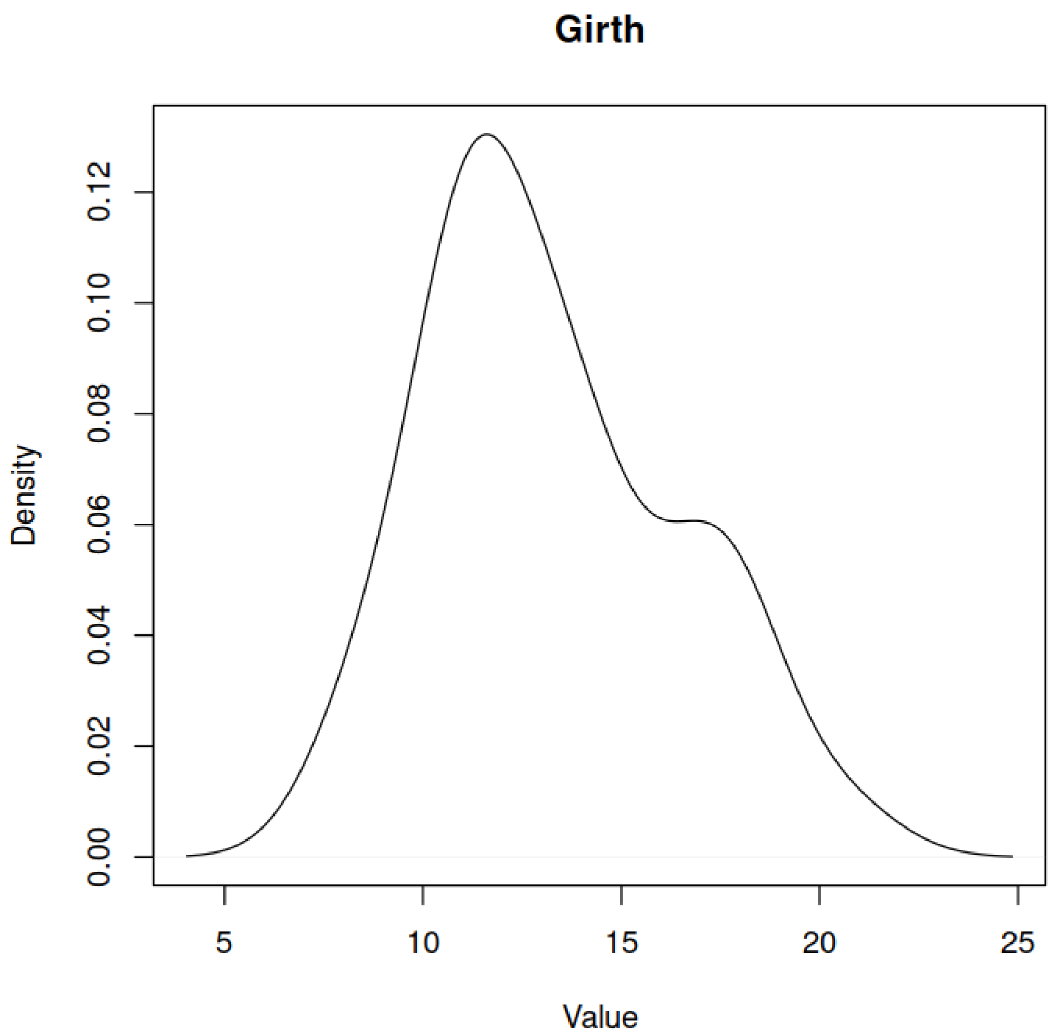

```
In [71]: hist.with.normal (trees$Volume)
```



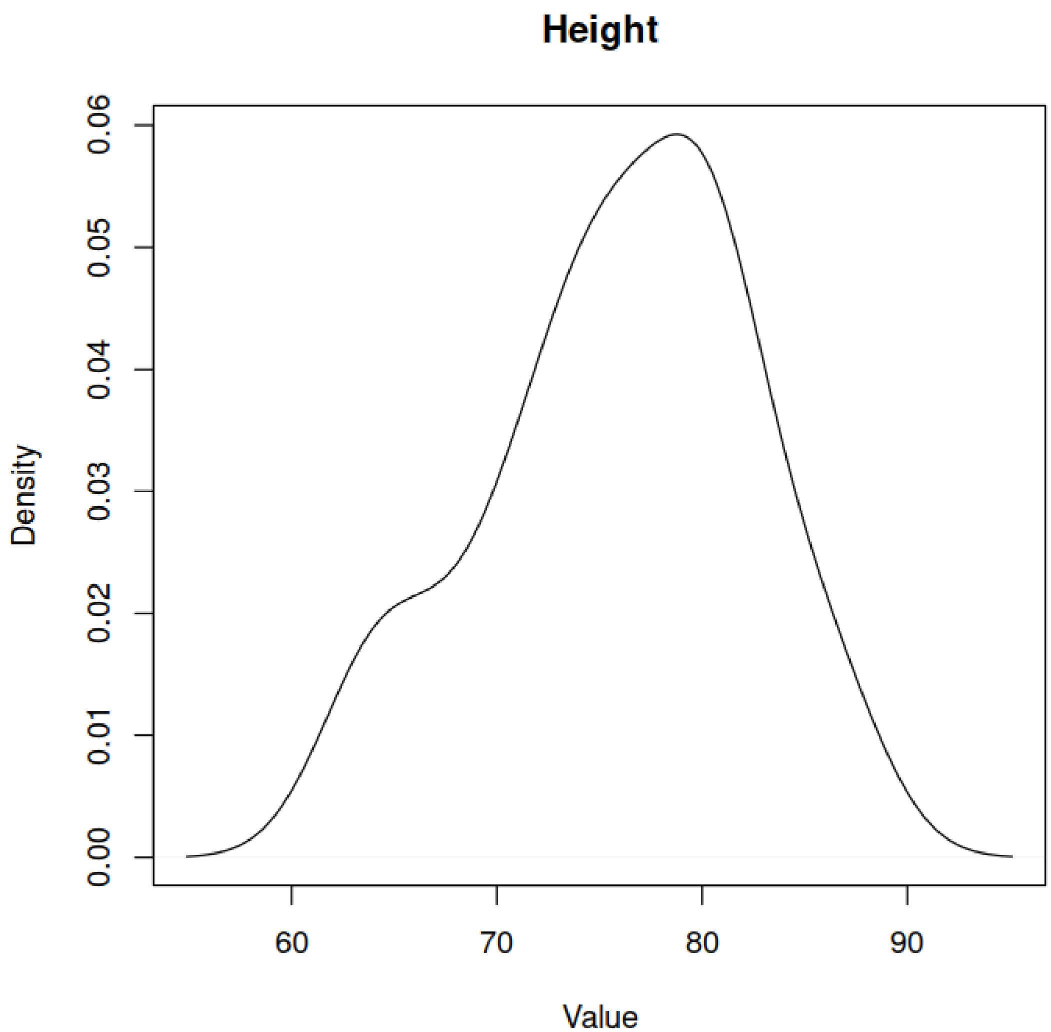

```
In [72]: plot(trees$Height, trees$Volume)
```



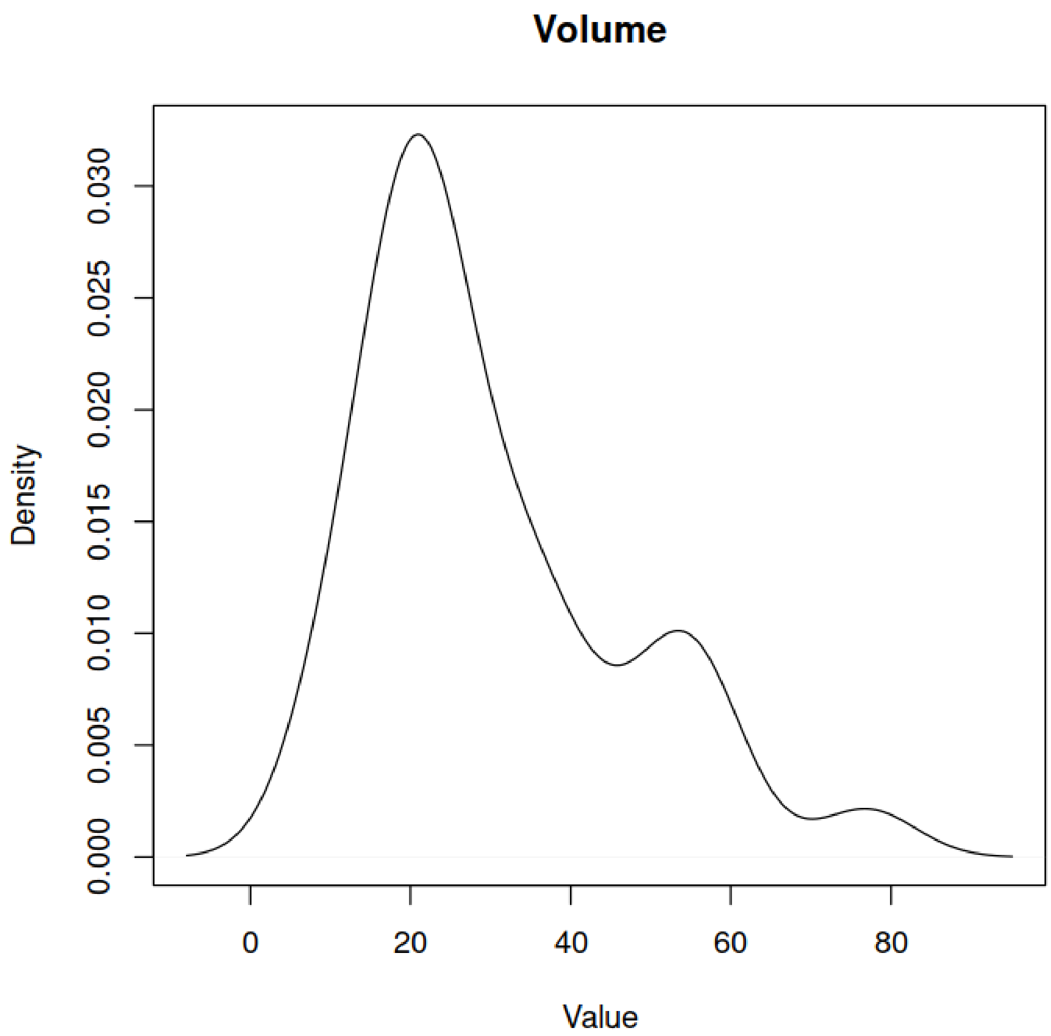

```
In [73]: plot(density(trees[,1]), xlab="Value", main=names(trees)[1])
```




```
In [74]: plot(density(trees[,2]), xlab="Value", main=names(trees)[2])
```



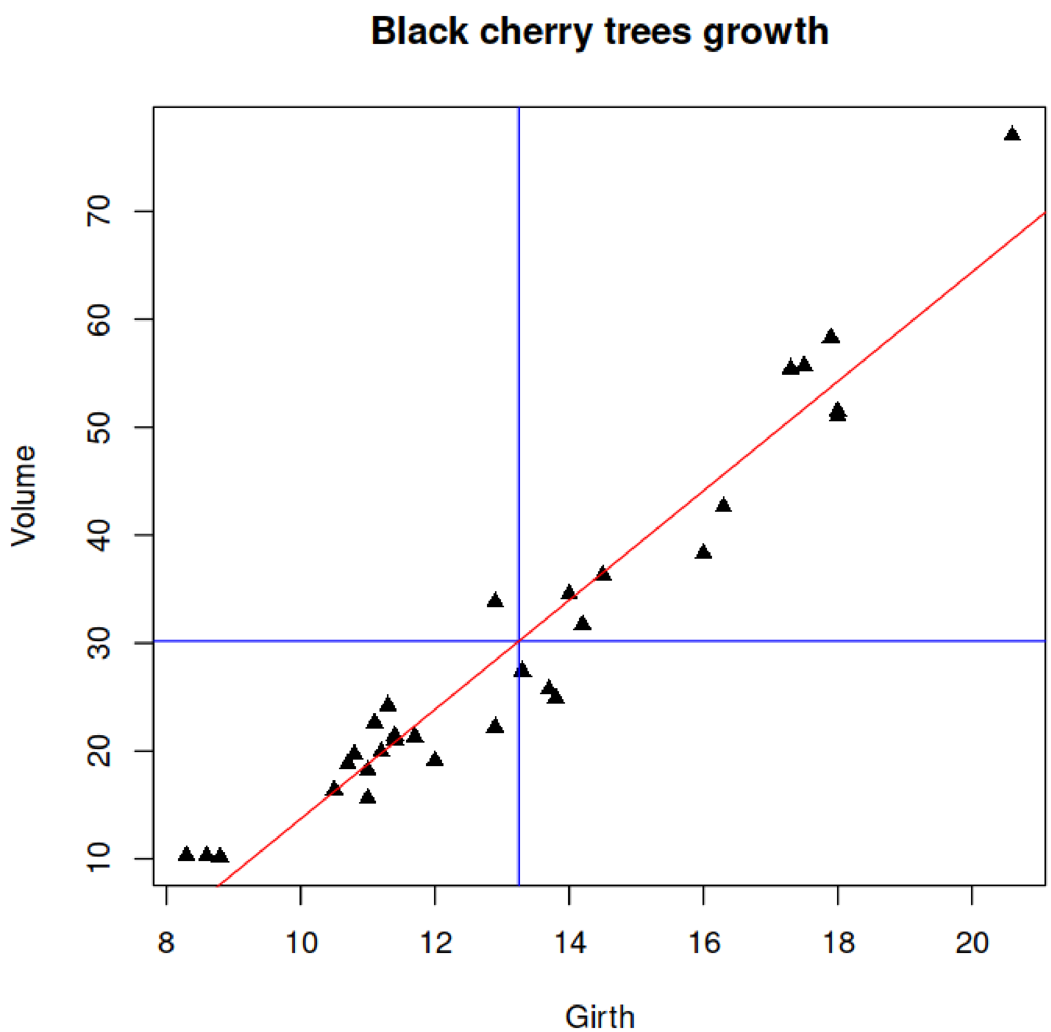

```
In [75]: plot(density(trees[,3]), xlab="Value", main=names(trees)[3])
```



Can we model Volume from Girth and Height?

```
In [76]: # 2D plot

         par(mfrow=c(1,1))  # sets a 1x1 display

         with(trees, plot(Girth,Volume,main = "Black cherry trees growth", pch = 17))  # elegant

         # add centered axes

         with(trees, abline(v = mean(Girth), col = "blue"))
         with(trees, abline(h = mean(Volume), col = "blue"))
```

```
# add regression line

with(trees, abline( lm(Volume ~ Girth), col = "red" ))
```



Individual variables can be directly named if we 'attach' the data frame to the current workspace

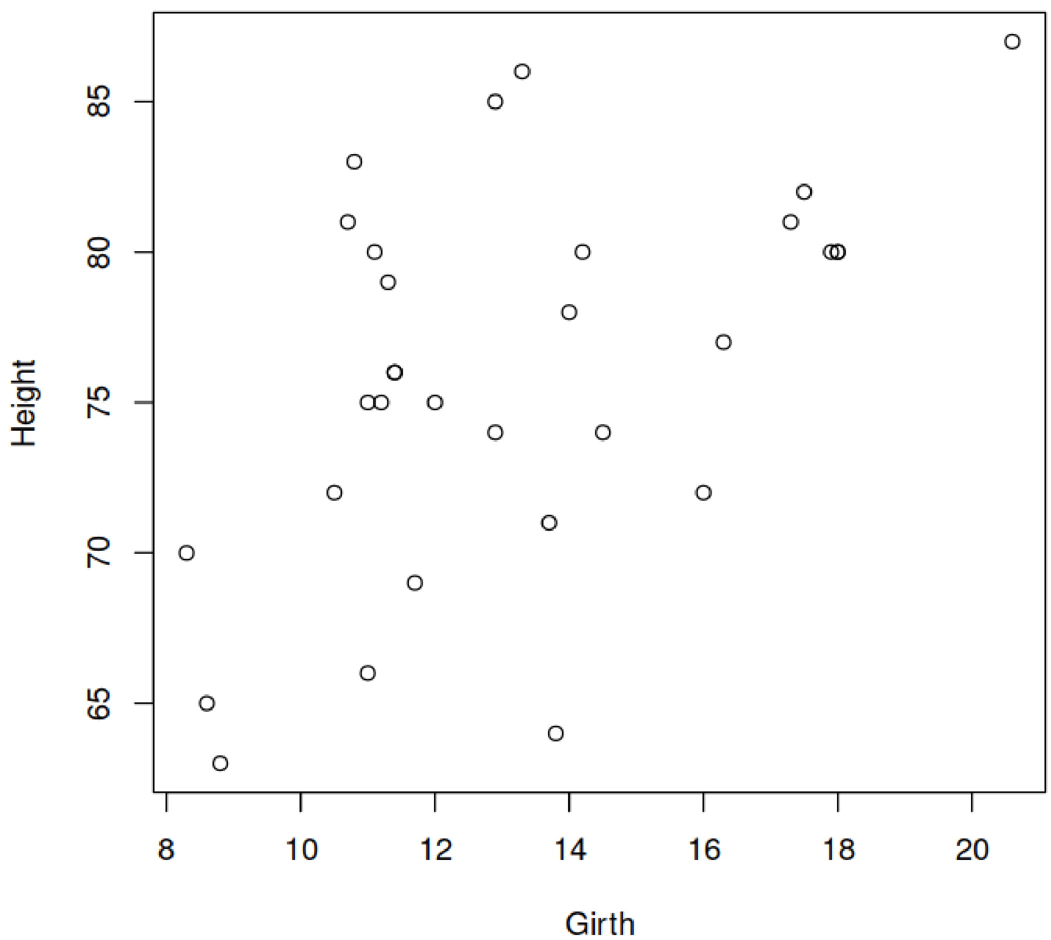
```
In [77]: attach(trees)
         Girth
```

1. 8.3 2. 8.6 3. 8.8 4. 10.5 5. 10.7 6. 10.8 7. 11 8. 11 9. 11.1 10. 11.2 11. 11.3 12. 11.4 13. 11.4 14. 11.7 15. 12 16. 12.9 17. 12.9 18. 13.3 19. 13.7 20. 13.8 21. 14 22. 14.2 23. 14.5 24. 16 25. 16.3 26. 17.3 27. 17.5 28. 17.9 29. 18 30. 18 31. 20.6

if we attach() we do not need to specify the dataframe all the time (handle this with care, if may mask errors)

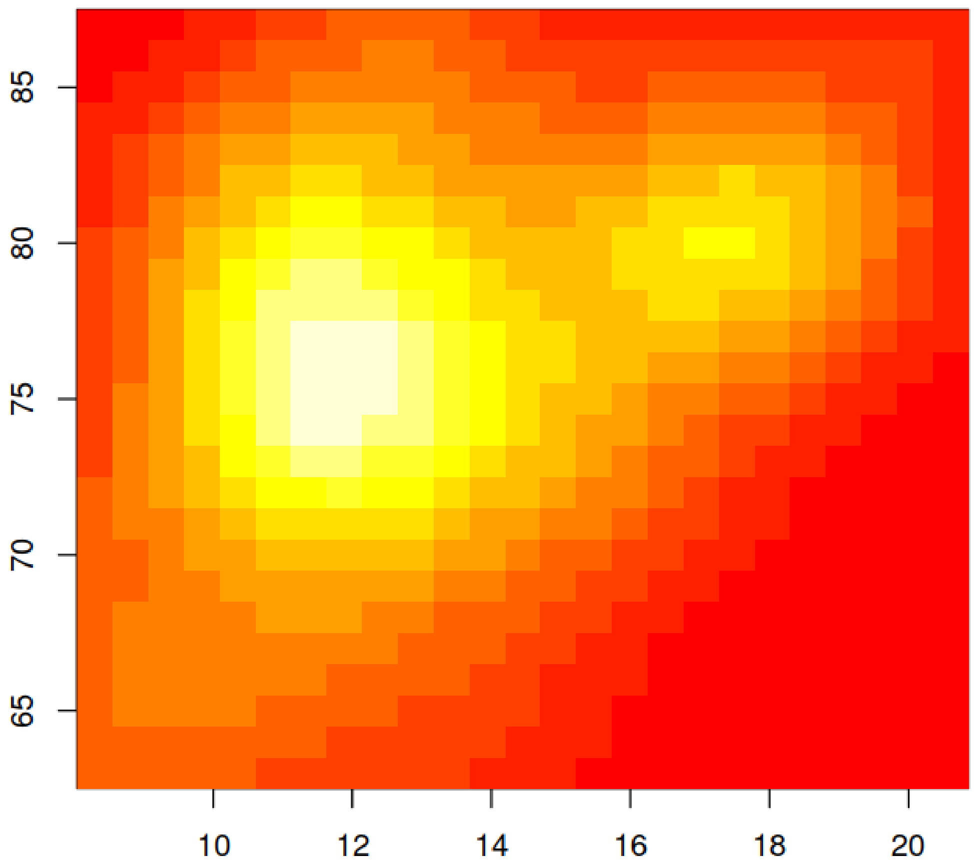
now 4 plots in one 2x2 device

```
In [78]: plot(Girth,Height)
```

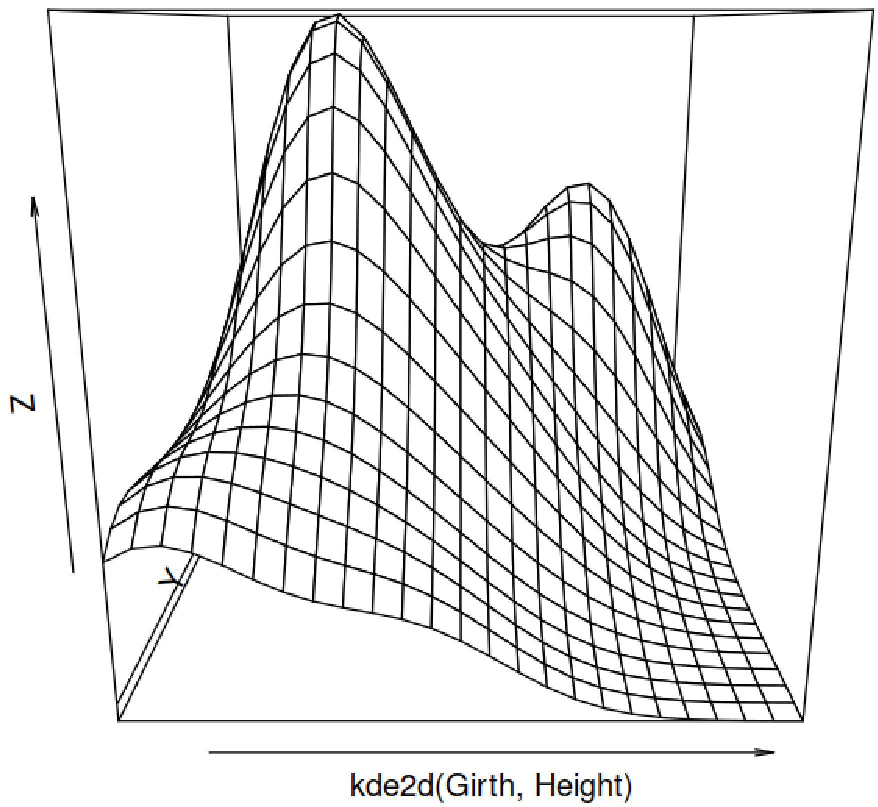


these are specially useful for spatial data, but can be applied:

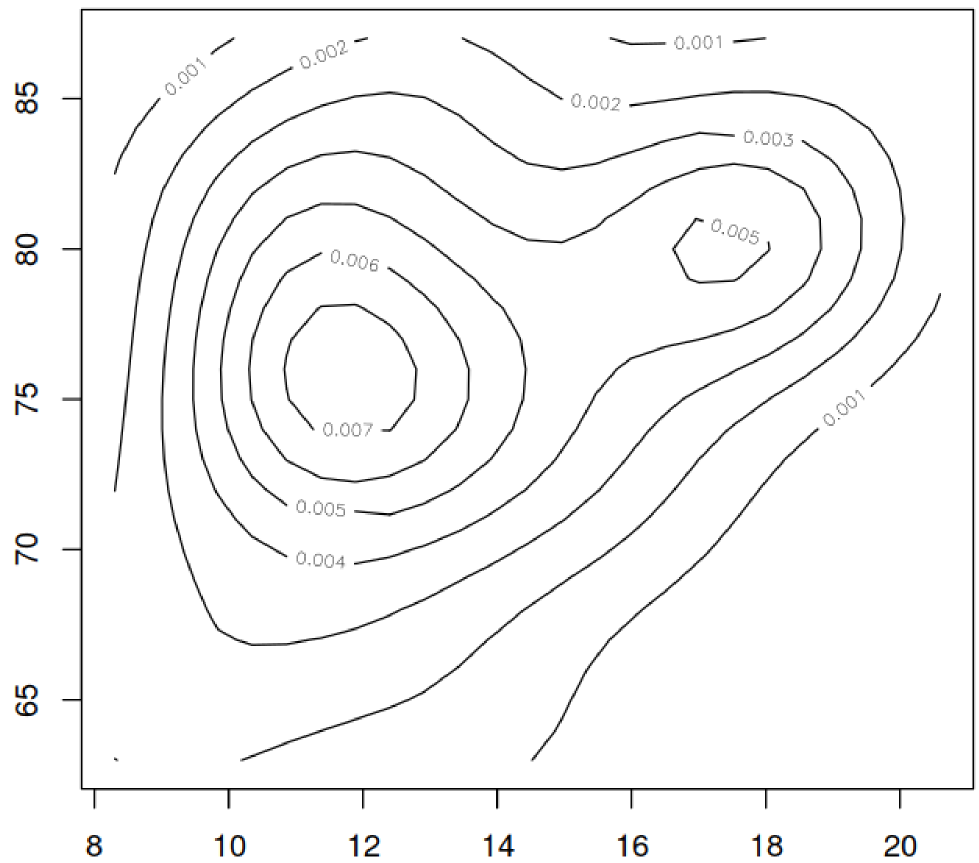
```
In [79]: library(MASS)

         image(kde2d(Girth,Height))          #colored image
```




```
In [80]: persp(kde2d(Girth,Height))          #perspective plot
```




```
In [81]: contour(kde2d(Girth,Height))        #contour plot
```



We can make more complicated (though still linear) models

```
In [82]: (model2 <- lm(log(Volume) ~ Girth + I(Girth^2)))

         (model3 <- lm(log(Volume) ~ poly(Girth, degree = 3)))


Call:
lm(formula = log(Volume) ~ Girth + I(Girth^2))

Coefficients:
(Intercept)        Girth   I(Girth^2)
   0.079824     0.318410    -0.005542
```

```
Call:
lm(formula = log(Volume) ~ poly(Girth, degree = 3))

Coefficients:
            (Intercept)  poly(Girth, degree = 3)1  poly(Girth, degree = 3)2
                 3.2727                    2.7942                   -0.3177
poly(Girth, degree = 3)3
                 0.1542
```

We can get (or set) the names of an object

```
In [83]: names(model2)
```

1. 'coefficients' 2. 'residuals' 3. 'effects' 4. 'rank' 5. 'fitted.values' 6. 'assign' 7. 'qr' 8. 'df.residual' 9. 'xlevels' 10. 'call' 11. 'terms' 12. 'model'

and its attributes

```
In [84]: attributes(model2)
         model2$fitted.values
```

$names 1. 'coefficients' 2. 'residuals' 3. 'effects' 4. 'rank' 5. 'fitted.values' 6. 'assign' 7. 'qr' 8. 'df.residual' 9. 'xlevels' 10. 'call' 11. 'terms' 12. 'model'

$class 'lm'

1 2.34085155180378 **2** 2.4082776063087 **3** 2.45267412944891 **4** 2.81214456656319 **5** 2.85232932274399 **6** 2.87225544687547 **7** 2.91177518722057 **8** 2.91177518722057 **9** 2.9313688034342 **10** 2.95085158367521 **11** 2.97022352794361 **12** 2.98948463623939 **13** 2.98948463623939 **14** 3.04660294529103 **15** 3.10272373058914 **16** 3.26510094396217 **17** 3.26510094396217 **18** 3.33438685906217 **19** 3.40189939860032 **20** 3.41850044355331 **21** 3.45137002554146 **22** 3.48379626363914 **23** 3.53160435099104 **24** 3.75568193144738 **25** 3.79750487627802 **26** 3.9297103541601 **27** 3.95482141806513 **28** 4.00371351420379 **29** 4.01565944830691 **30** 4.01565944830691 **31** 4.28735030859993

or display the internal structure of an object

```
In [85]: str(trees)
```

```
'data.frame':	31 obs. of  3 variables:
 $ Girth : num  8.3 8.6 8.8 10.5 10.7 10.8 11 11 11.1 11.2 ...
 $ Height: num  70 65 63 72 81 83 66 75 80 75 ...
 $ Volume: num  10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 ...
```

In [86]: str(model2)

```
List of 12
 $ coefficients : Named num [1:3] 0.07982 0.31841 -0.00554
  ..- attr(*, "names")= chr [1:3] "(Intercept)" "Girth" "I(Girth^2)"
 $ residuals    : Named num [1:31] -0.00871 -0.07613 -0.13029 -0.01486 0.08153 ...
  ..- attr(*, "names")= chr [1:31] "1" "2" "3" "4" ...
 $ effects      : Named num [1:31] -18.2218 2.7942 0.3177 -0.0108 0.0828 ...
  ..- attr(*, "names")= chr [1:31] "(Intercept)" "Girth" "I(Girth^2)" "" ...
 $ rank         : int 3
 $ fitted.values: Named num [1:31] 2.34 2.41 2.45 2.81 2.85 ...
  ..- attr(*, "names")= chr [1:31] "1" "2" "3" "4" ...
 $ assign       : int [1:3] 0 1 2
 $ qr           :List of 5
  ..$ qr   : num [1:31, 1:3] -5.57 0.18 0.18 0.18 0.18 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:31] "1" "2" "3" "4" ...
  .. .. ..$ : chr [1:3] "(Intercept)" "Girth" "I(Girth^2)"
  .. ..- attr(*, "assign")= int [1:3] 0 1 2
  ..$ qraux: num [1:3] 1.18 1.23 1.19
  ..$ pivot: int [1:3] 1 2 3
  ..$ tol  : num 1e-07
  ..$ rank : int 3
  ..- attr(*, "class")= chr "qr"
 $ df.residual  : int 28
 $ xlevels      : Named list()
 $ call         : language lm(formula = log(Volume) ~ Girth + I(Girth^2))
 $ terms        :Classes 'terms', 'formula'  language log(Volume) ~ Girth + I(Girth^2)
  .. ..- attr(*, "variables")= language list(log(Volume), Girth, I(Girth^2))
  .. ..- attr(*, "factors")= int [1:3, 1:2] 0 1 0 0 0 1
  .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. ..$ : chr [1:3] "log(Volume)" "Girth" "I(Girth^2)"
  .. .. .. ..$ : chr [1:2] "Girth" "I(Girth^2)"
  .. ..- attr(*, "term.labels")= chr [1:2] "Girth" "I(Girth^2)"
  .. ..- attr(*, "order")= int [1:2] 1 1
  .. ..- attr(*, "intercept")= int 1
  .. ..- attr(*, "response")= int 1
  .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
  .. ..- attr(*, "predvars")= language list(log(Volume), Girth, I(Girth^2))
  .. ..- attr(*, "dataClasses")= Named chr [1:3] "numeric" "numeric" "numeric"
  .. .. ..- attr(*, "names")= chr [1:3] "log(Volume)" "Girth" "I(Girth^2)"
 $ model        :'data.frame':       31 obs. of  3 variables:
  ..$ log(Volume): num [1:31] 2.33 2.33 2.32 2.8 2.93 ...
  ..$ Girth      : num [1:31] 8.3 8.6 8.8 10.5 10.7 10.8 11 11 11.1 11.2 ...
  ..$ I(Girth^2) : 'AsIs' num [1:31]  68.89  73.96  77.44 110.25 114.49 ...
  ..- attr(*, "terms")=Classes 'terms', 'formula'  language log(Volume) ~ Girth + I(Girth^2)
  .. .. ..- attr(*, "variables")= language list(log(Volume), Girth, I(Girth^2))
  .. .. ..- attr(*, "factors")= int [1:3, 1:2] 0 1 0 0 0 1
```

```
  .. .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. .. ..$ : chr [1:3] "log(Volume)" "Girth" "I(Girth^2)"
  .. .. .. .. ..$ : chr [1:2] "Girth" "I(Girth^2)"
  .. .. ..- attr(*, "term.labels")= chr [1:2] "Girth" "I(Girth^2)"
  .. .. ..- attr(*, "order")= int [1:2] 1 1
  .. .. ..- attr(*, "intercept")= int 1
  .. .. ..- attr(*, "response")= int 1
  .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
  .. .. ..- attr(*, "predvars")= language list(log(Volume), Girth, I(Girth^2))
  .. .. ..- attr(*, "dataClasses")= Named chr [1:3] "numeric" "numeric" "numeric"
  .. .. .. ..- attr(*, "names")= chr [1:3] "log(Volume)" "Girth" "I(Girth^2)"
 - attr(*, "class")= chr "lm"
```

```
In [87]: unclass(model2)
```

```
$coefficients
 (Intercept)        Girth   I(Girth^2)
 0.079823791  0.318409912 -0.005541799

$residuals
           1            2            3            4            5            6
-0.008707657 -0.076133711 -0.130286409 -0.014863232  0.081527547  0.108363189
           7            8            9           10           11           12
-0.164504273 -0.010353593  0.186581103  0.039868148  0.216129105  0.055037801
          13           14           15           16           17           18
 0.073906286  0.012104127 -0.153035396 -0.165008655  0.255359859 -0.023843846
          19           20           21           22           23           24
-0.155408407 -0.203632640  0.089589298 -0.027479583  0.060213390 -0.110232035
          25           26           27           28           29           30
-0.045650623  0.084869240  0.065158729  0.061888579 -0.074077641 -0.083833816
          31
 0.056455113

$effects
  (Intercept)         Girth    I(Girth^2)
-18.221799052   2.794229219   0.317692907  -0.010753221   0.082844985

  0.108335088  -0.167121951  -0.012971272   0.182719384   0.034796222

  0.209880804   0.047646958   0.066515443   0.001488653  -0.166571007

 -0.185477719   0.234890795  -0.046514796  -0.179739921  -0.228294714

  0.064367603  -0.053125569   0.034184712  -0.133606771  -0.067581090

  0.069952122   0.052050273   0.052803437  -0.082072373  -0.091828548
```

```
  0.088686286

$rank
[1] 3

$fitted.values
       1        2        3        4        5        6        7        8 
2.340852 2.408278 2.452674 2.812145 2.852329 2.872255 2.911775 2.911775 
       9       10       11       12       13       14       15       16 
2.931369 2.950852 2.970224 2.989485 2.989485 3.046603 3.102724 3.265101 
      17       18       19       20       21       22       23       24 
3.265101 3.334387 3.401899 3.418500 3.451370 3.483796 3.531604 3.755682 
      25       26       27       28       29       30       31 
3.797505 3.929710 3.954821 4.003714 4.015659 4.015659 4.287350 

$assign
[1] 0 1 2

$qr
$qr
   (Intercept)        Girth    I(Girth^2)
1   -5.5677644 -73.76389754 -1.030315e+03
2    0.1796053  17.18829309  4.833618e+02
3    0.1796053   0.21496908 -5.732668e+01
4    0.1796053   0.11606455 -4.428254e-02
5    0.1796053   0.10442873 -6.576147e-02
6    0.1796053   0.09861081 -7.597761e-02
7    0.1796053   0.08697499 -9.536327e-02
8    0.1796053   0.08697499 -9.536327e-02
9    0.1796053   0.08115707 -1.045328e-01
10   0.1796053   0.07533916 -1.133534e-01
11   0.1796053   0.06952125 -1.218252e-01
12   0.1796053   0.06370333 -1.299481e-01
13   0.1796053   0.06370333 -1.299481e-01
14   0.1796053   0.04624959 -1.522234e-01
15   0.1796053   0.02879585 -1.713589e-01
16   0.1796053  -0.02356537 -2.099259e-01
17   0.1796053  -0.02356537 -2.099259e-01
18   0.1796053  -0.04683702 -2.179960e-01
19   0.1796053  -0.07010867 -2.204840e-01
20   0.1796053  -0.07592659 -2.202338e-01
21   0.1796053  -0.08756241 -2.186868e-01
22   0.1796053  -0.09919824 -2.157443e-01
23   0.1796053  -0.11665198 -2.087140e-01
24   0.1796053  -0.20392068 -1.264637e-01
25   0.1796053  -0.22137442 -1.005939e-01
26   0.1796053  -0.27955356  8.315626e-03
27   0.1796053  -0.29118938  3.428407e-02
```

```
28   0.1796053  -0.31446104  9.040749e-02
29   0.1796053  -0.32027895  1.053105e-01
30   0.1796053  -0.32027895  1.053105e-01
31   0.1796053  -0.47154470  6.152459e-01
attr(,"assign")
[1] 0 1 2

$qraux
[1] 1.179605 1.226605 1.194632

$pivot
[1] 1 2 3

$tol
[1] 1e-07

$rank
[1] 3

attr(,"class")
[1] "qr"

$df.residual
[1] 28

$xlevels
named list()

$call
lm(formula = log(Volume) ~ Girth + I(Girth^2))

$terms
log(Volume) ~ Girth + I(Girth^2)
attr(,"variables")
list(log(Volume), Girth, I(Girth^2))
attr(,"factors")
            Girth I(Girth^2)
log(Volume)     0          0
Girth           1          0
I(Girth^2)      0          1
attr(,"term.labels")
[1] "Girth"      "I(Girth^2)"
attr(,"order")
[1] 1 1
attr(,"intercept")
[1] 1
attr(,"response")
[1] 1
```

```
attr(,".Environment")
<environment: R_GlobalEnv>
attr(,"predvars")
list(log(Volume), Girth, I(Girth^2))
attr(,"dataClasses")
log(Volume)       Girth  I(Girth^2)
  "numeric"   "numeric"   "numeric"

$model
   log(Volume) Girth I(Girth^2)
1     2.332144   8.3      68.89
2     2.332144   8.6      73.96
3     2.322388   8.8      77.44
4     2.797281  10.5     110.25
5     2.933857  10.7     114.49
6     2.980619  10.8     116.64
7     2.747271  11.0        121
8     2.901422  11.0        121
9     3.117950  11.1     123.21
10    2.990720  11.2     125.44
11    3.186353  11.3     127.69
12    3.044522  11.4     129.96
13    3.063391  11.4     129.96
14    3.058707  11.7     136.89
15    2.949688  12.0        144
16    3.100092  12.9     166.41
17    3.520461  12.9     166.41
18    3.310543  13.3     176.89
19    3.246491  13.7     187.69
20    3.214868  13.8     190.44
21    3.540959  14.0        196
22    3.456317  14.2     201.64
23    3.591818  14.5     210.25
24    3.645450  16.0        256
25    3.751854  16.3     265.69
26    4.014580  17.3     299.29
27    4.019980  17.5     306.25
28    4.065602  17.9     320.41
29    3.941582  18.0        324
30    3.931826  18.0        324
31    4.343805  20.6     424.36
```

though in general summary will be enough

```
In [88]: summary(model3)
         detach(trees)
```

```
Call:
lm(formula = log(Volume) ~ poly(Girth, degree = 3))

Residuals:
     Min       1Q   Median       3Q      Max
-0.20146 -0.05656 -0.01399  0.07934  0.24525

Coefficients:
                         Estimate Std. Error t value Pr(>|t|)
(Intercept)                3.2727     0.0212 154.354   <2e-16 ***
poly(Girth, degree = 3)1   2.7942     0.1181  23.669   <2e-16 ***
poly(Girth, degree = 3)2  -0.3177     0.1181  -2.691   0.0121 *
poly(Girth, degree = 3)3   0.1542     0.1181   1.306   0.2026
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.1181 on 27 degrees of freedom
Multiple R-squared:  0.9547,Adjusted R-squared:  0.9497
F-statistic: 189.7 on 3 and 27 DF,  p-value: < 2.2e-16
```

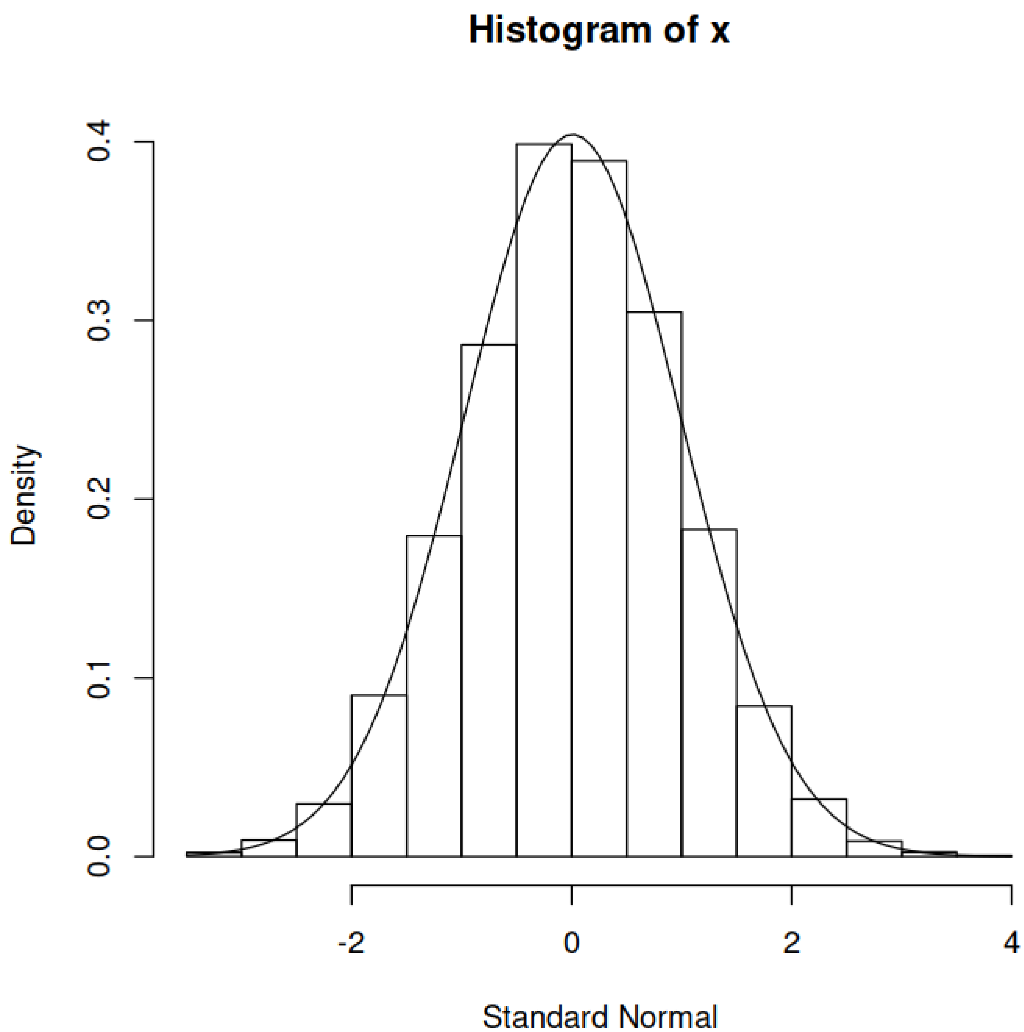
1.17 Creating random samples

Let's create different iid random samples: Normal (Gaussian), uniform, Chi-square and Weibull; we show a histogram (left) and a density estimation (right) against the normal pdf

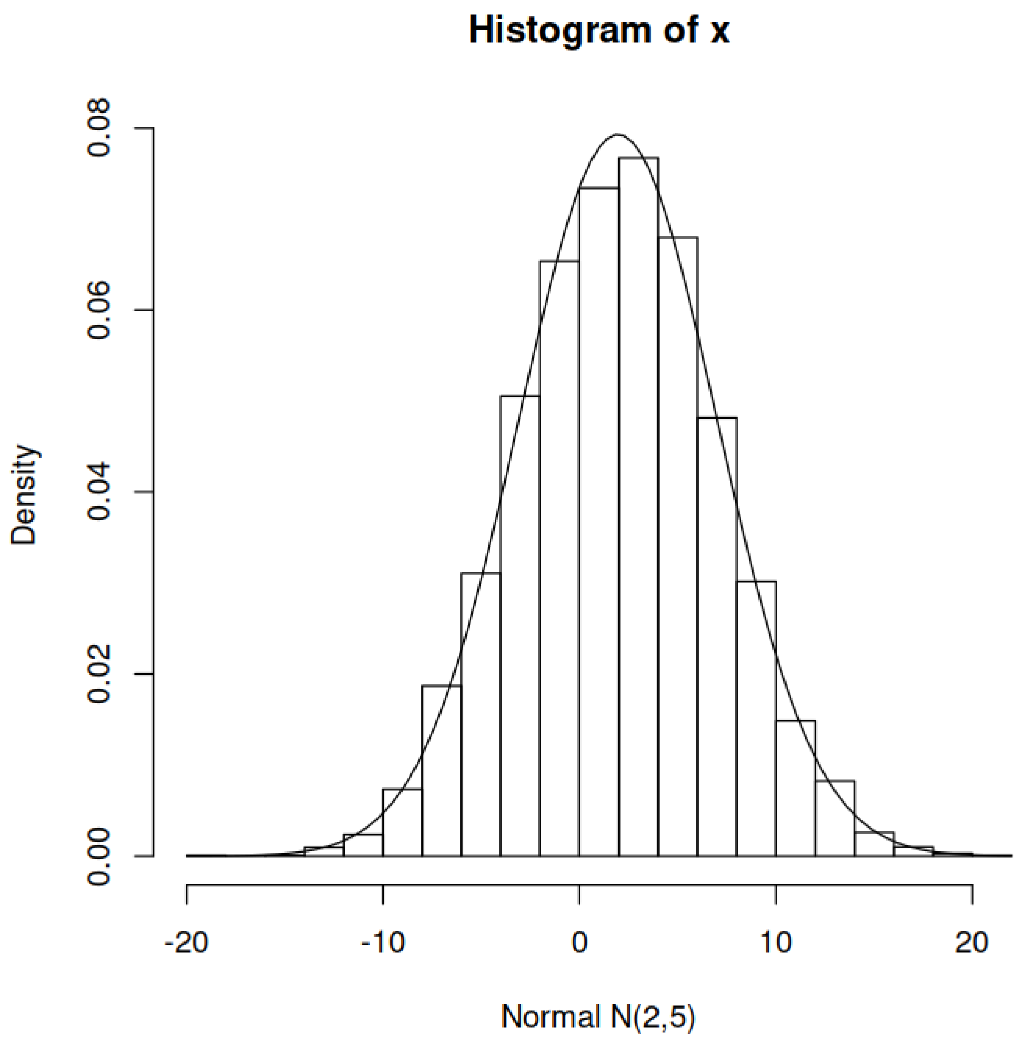
In [89]:
```
N <- 10000

par(mfrow=c(2,3))
```

In [90]:
```
# Normal (Gaussian) distribution

nd1 <- rnorm(N, mean = 0, sd = 1)
nd2 <- rnorm(N, mean = 2, sd = 5)
nd3 <- rnorm(N, mean = -10, sd = 0.01)
```

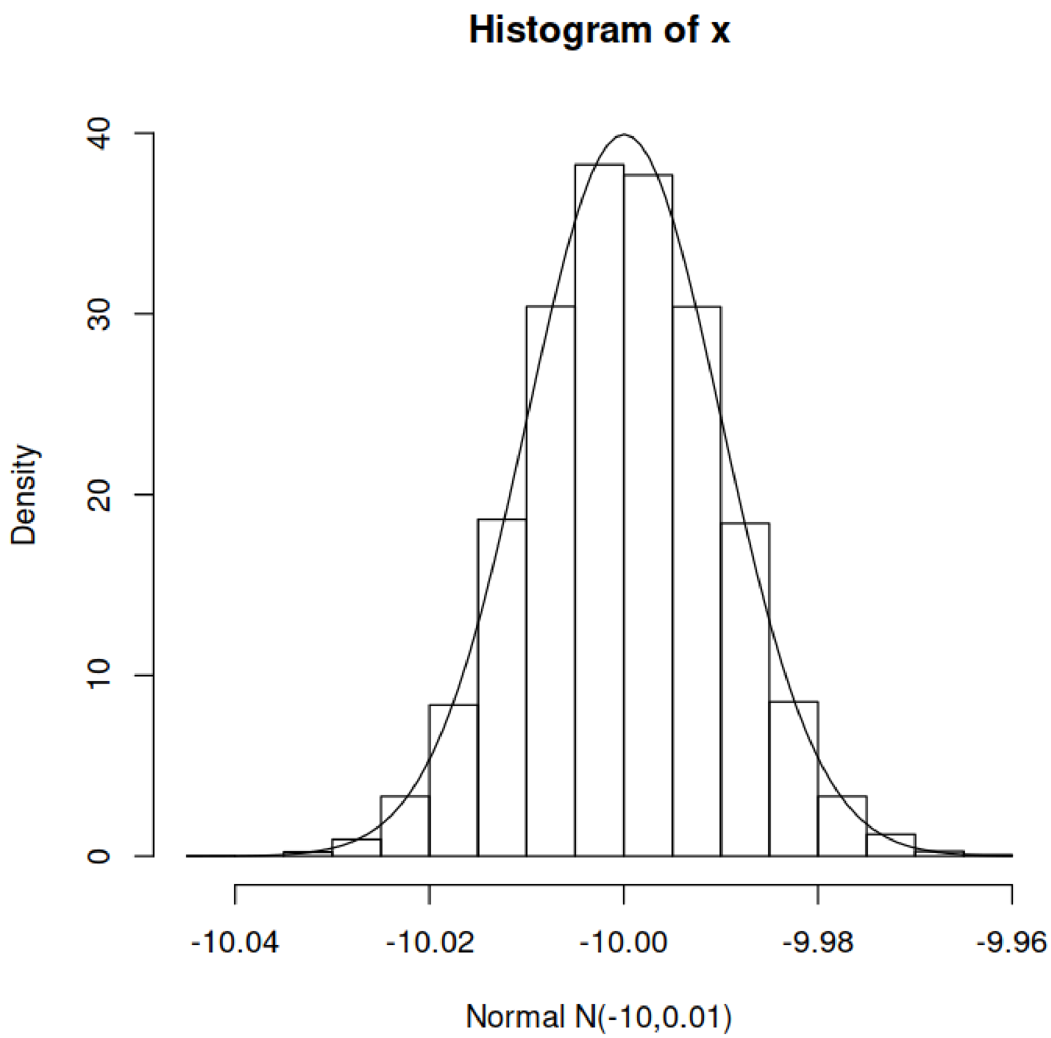
In [91]:
```
hist.with.normal (nd1, "Standard Normal")
```



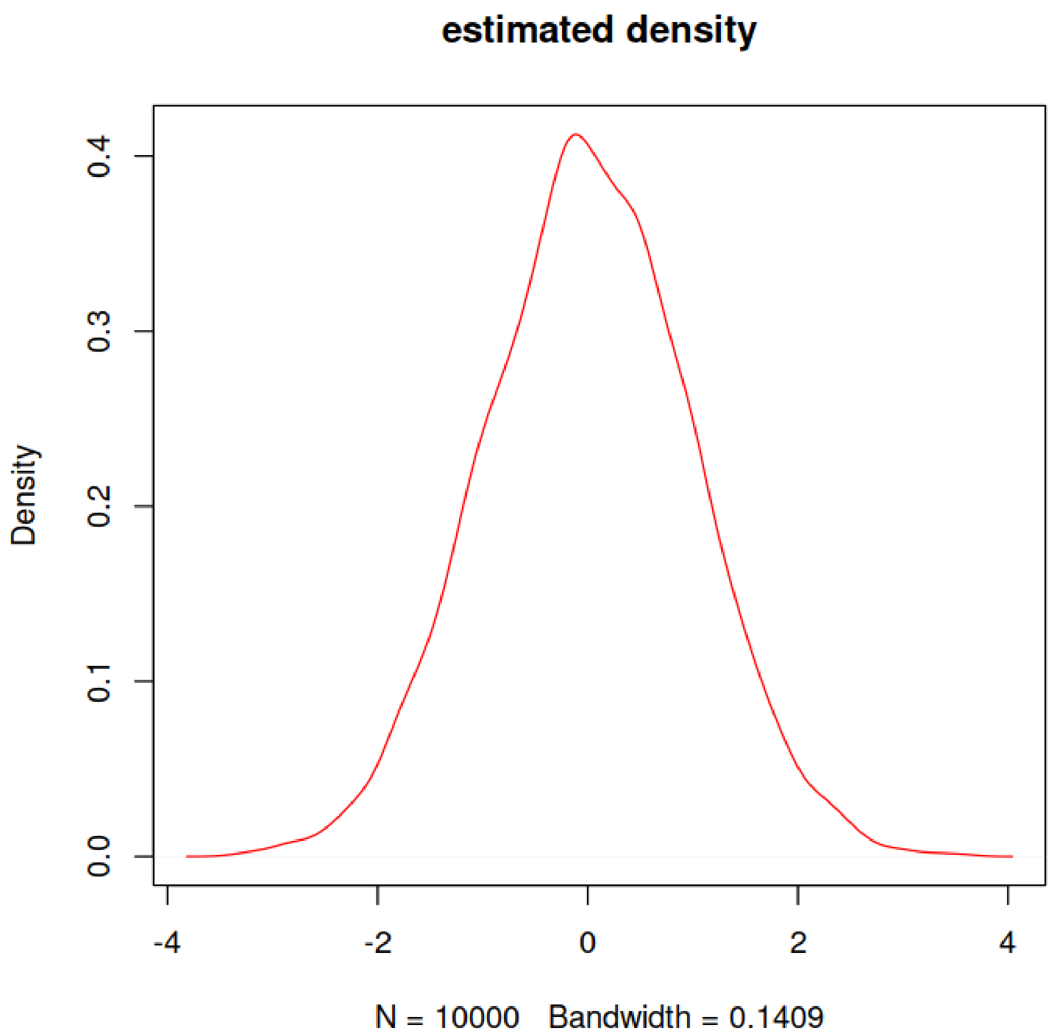

```
In [92]: hist.with.normal (nd2, "Normal N(2,5)")
```



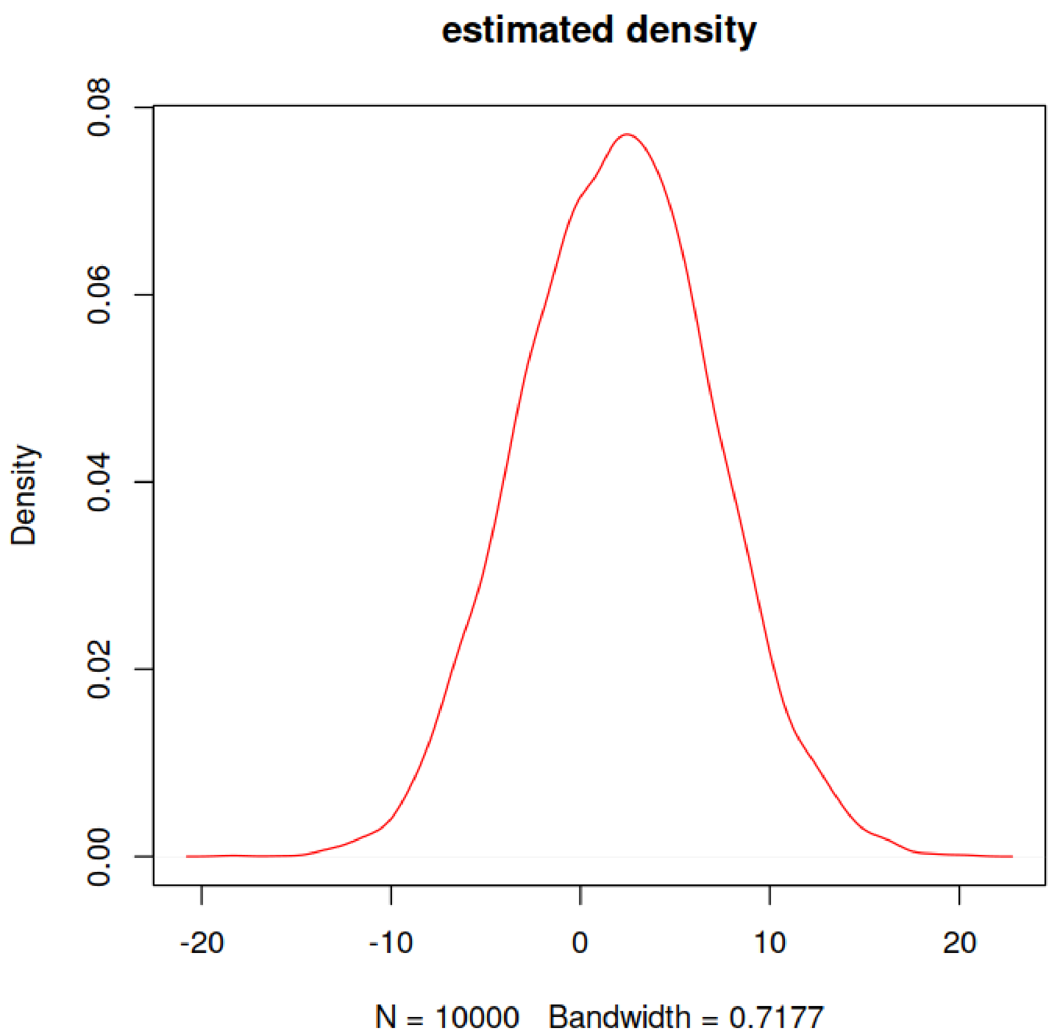

```
In [93]: hist.with.normal (nd3, "Normal N(-10,0.01)")
```



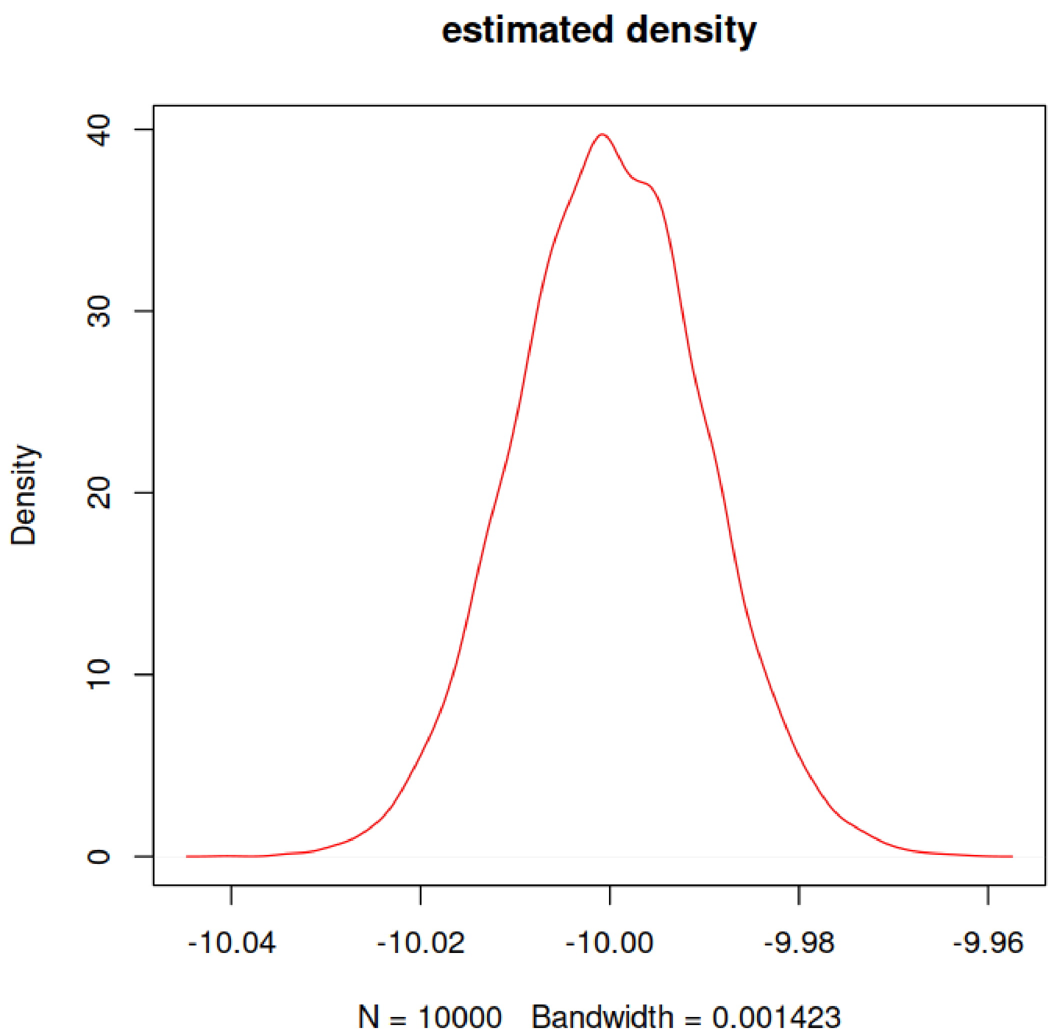

```
In [94]: plot(density(nd1), col='red', main="estimated density")
```



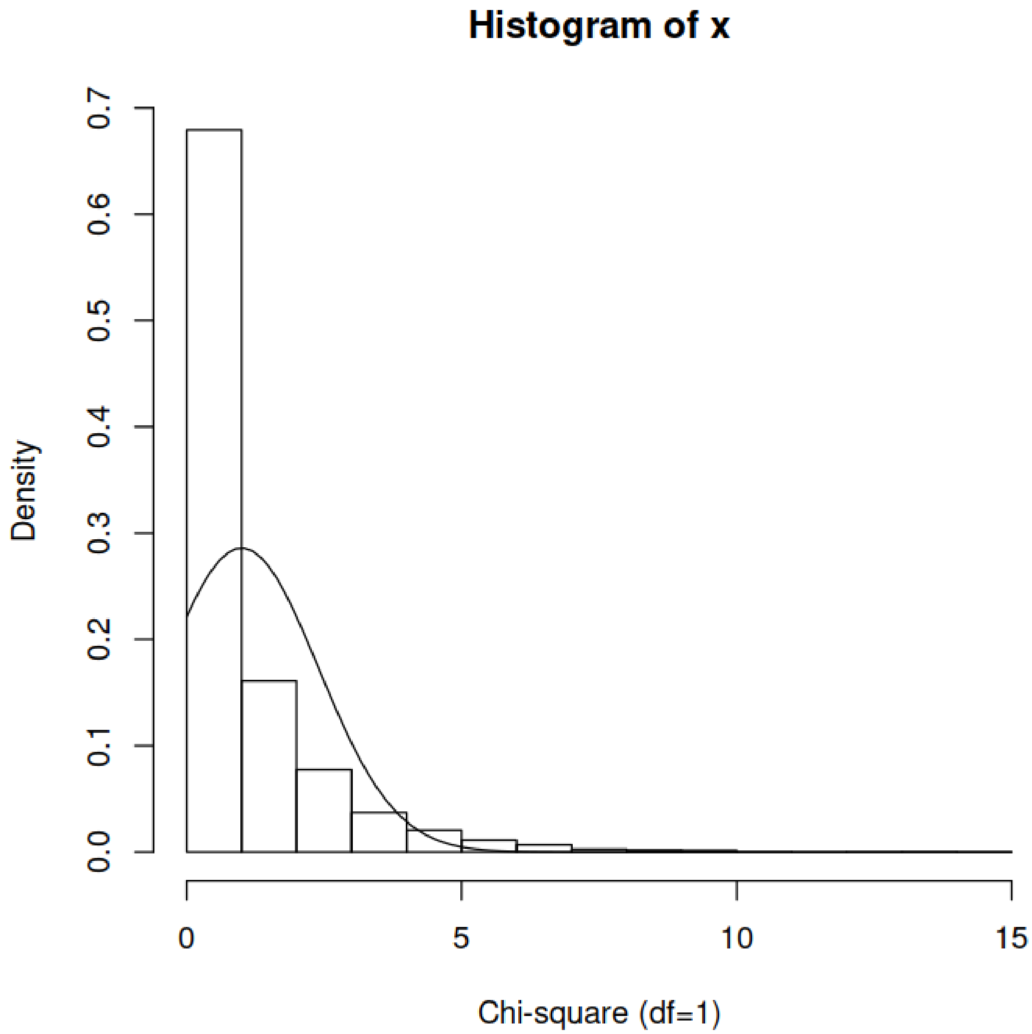

```
In [95]: plot(density(nd2), col='red', main="estimated density")
```



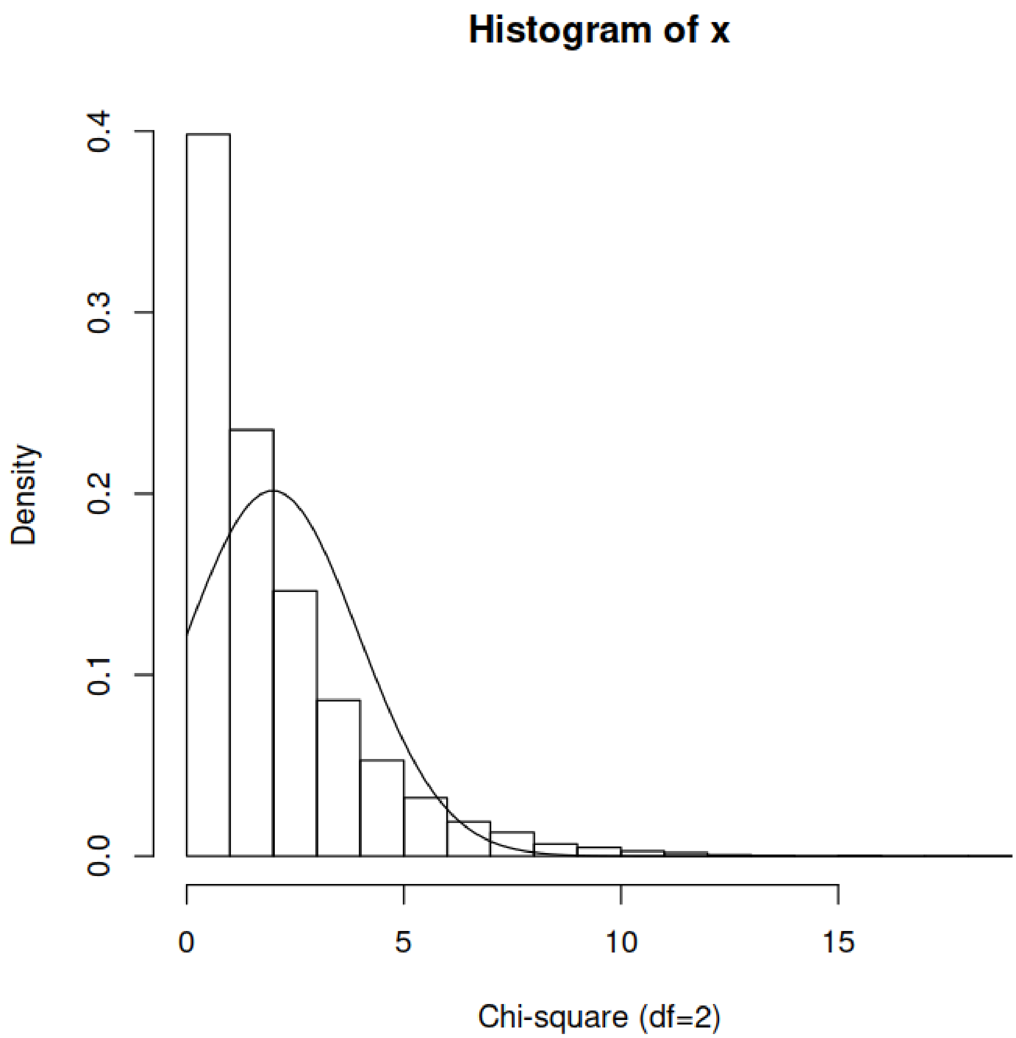

```
In [96]: plot(density(nd3), col='red', main="estimated density")
```



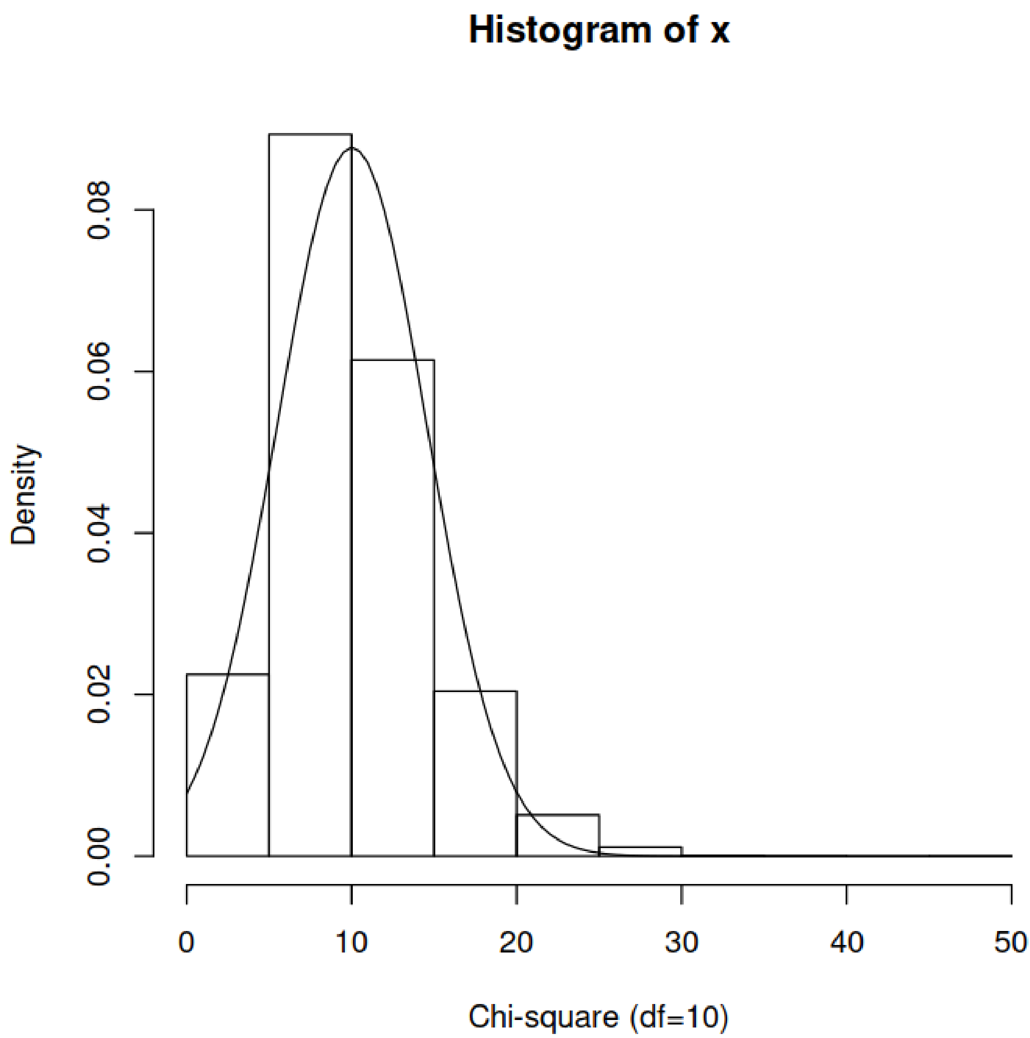

```
In [97]: # Chi-square distribution

         cq1 <- rchisq(N, 1)
         cq2 <- rchisq(N, 2)
         cq3 <- rchisq(N, 10)

In [98]: hist.with.normal (cq1, "Chi-square (df=1)")
```



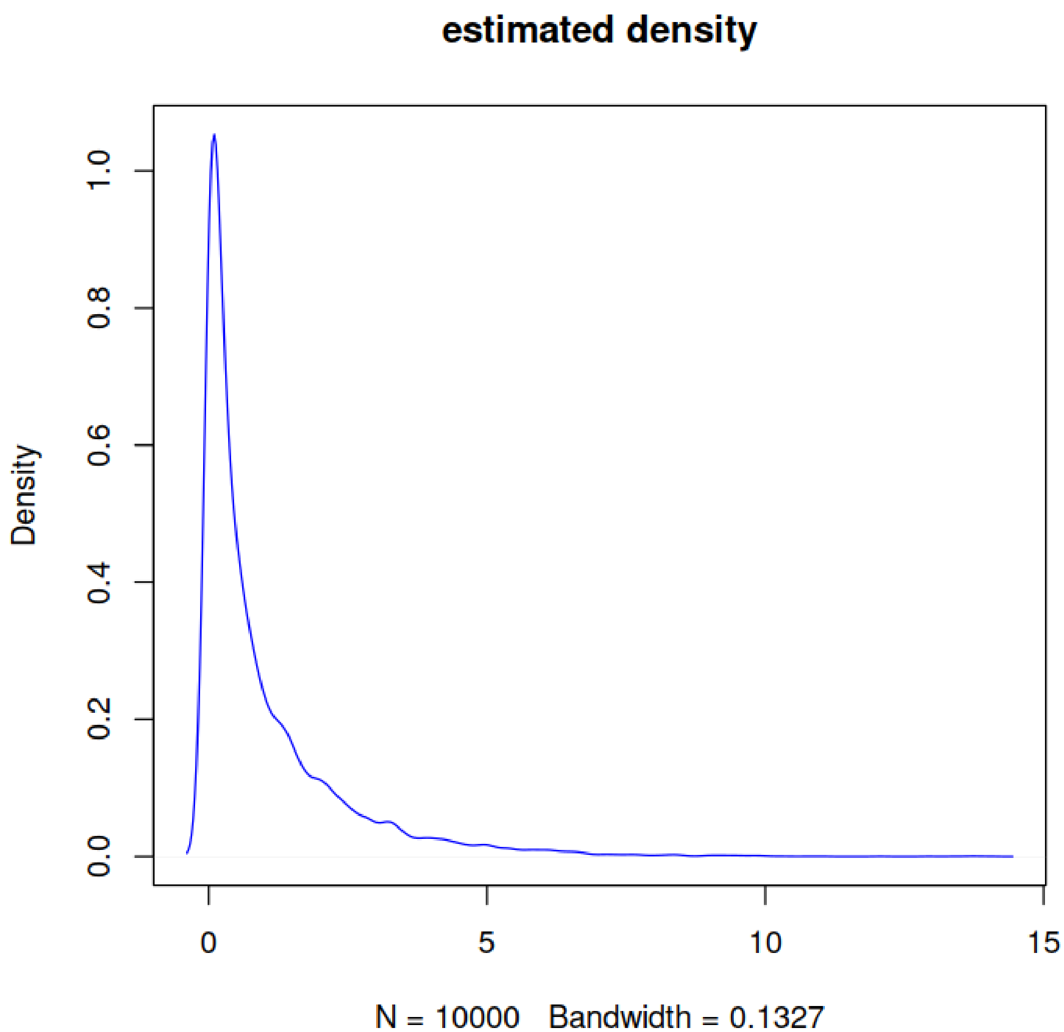

```
In [99]: hist.with.normal (cq2, "Chi-square (df=2)")
```



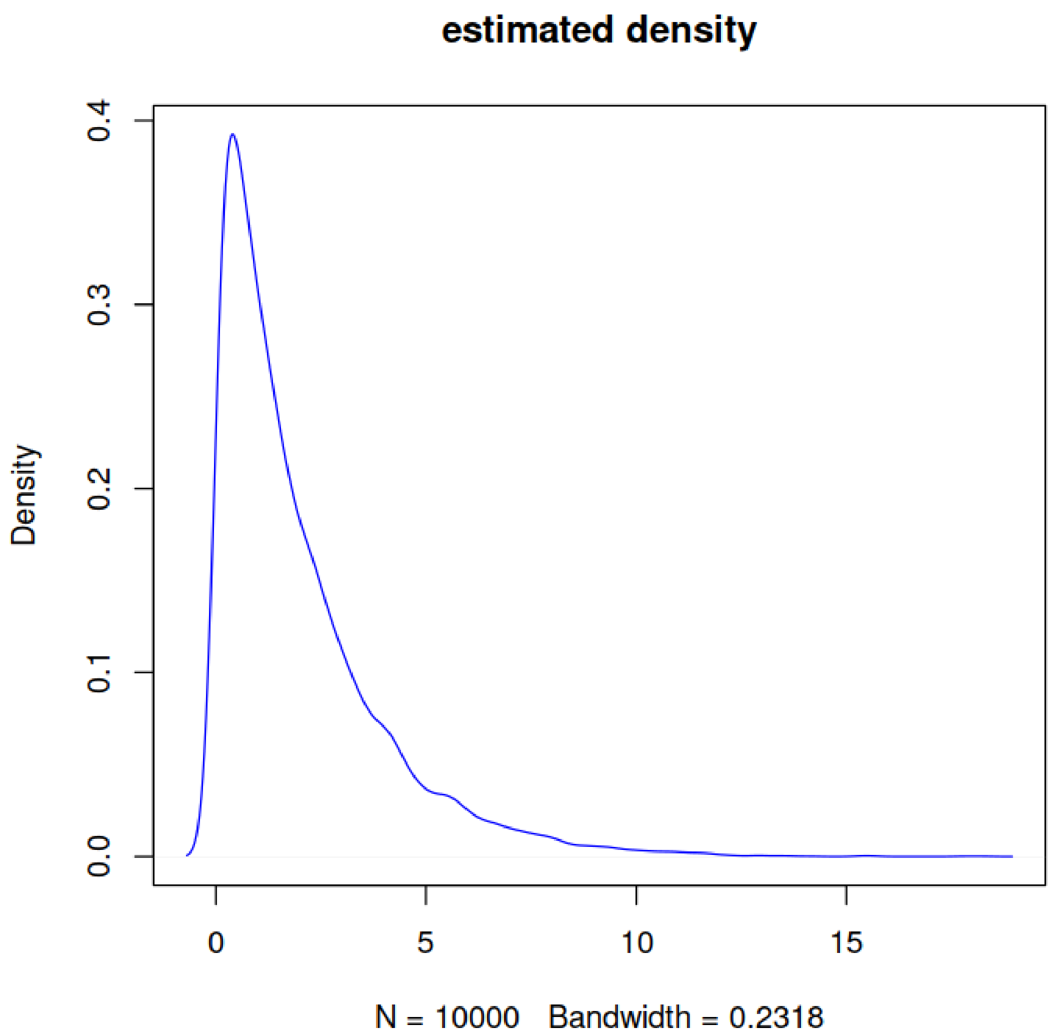

```
In [100]: hist.with.normal (cq3, "Chi-square (df=10)")
```



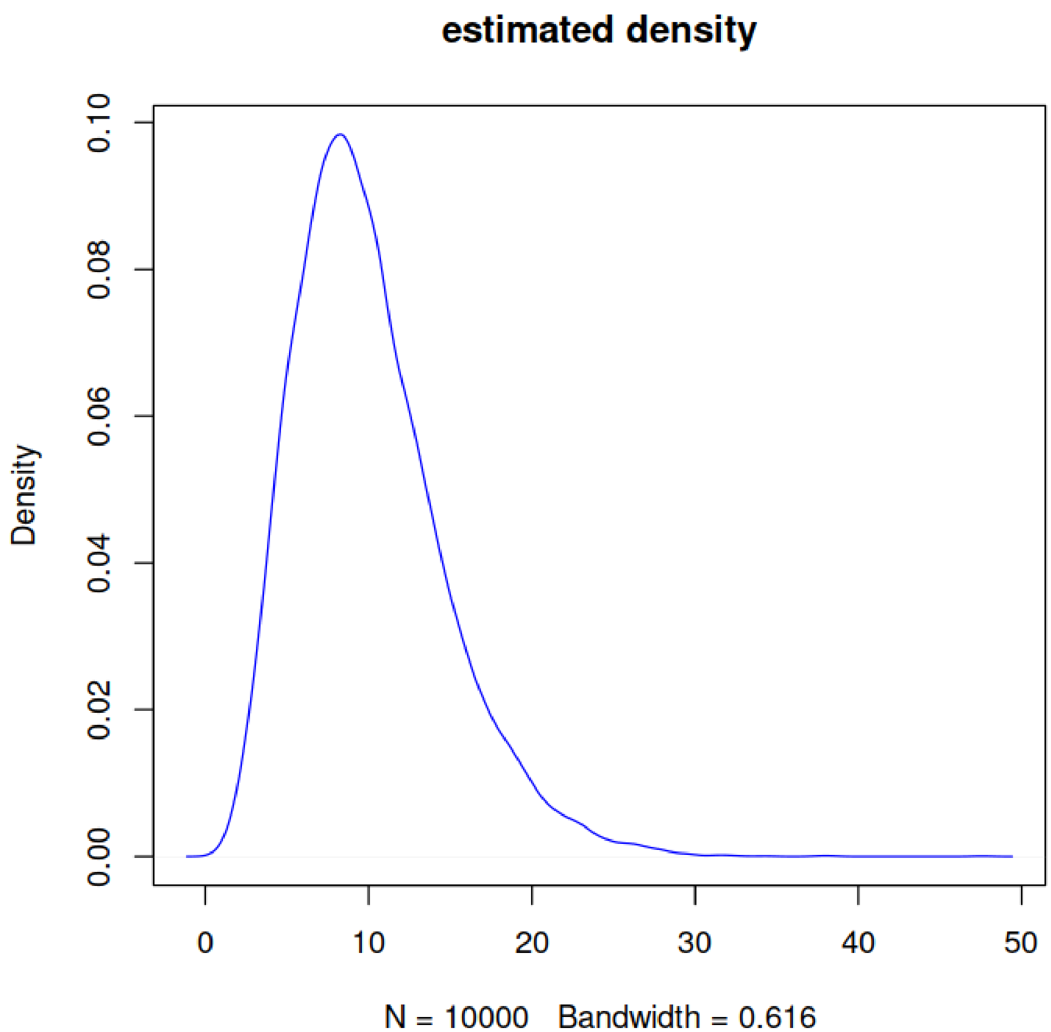

```
In [101]: plot(density(cq1), col='blue', main="estimated density")
```




```
In [102]: plot(density(cq2), col='blue', main="estimated density")
```

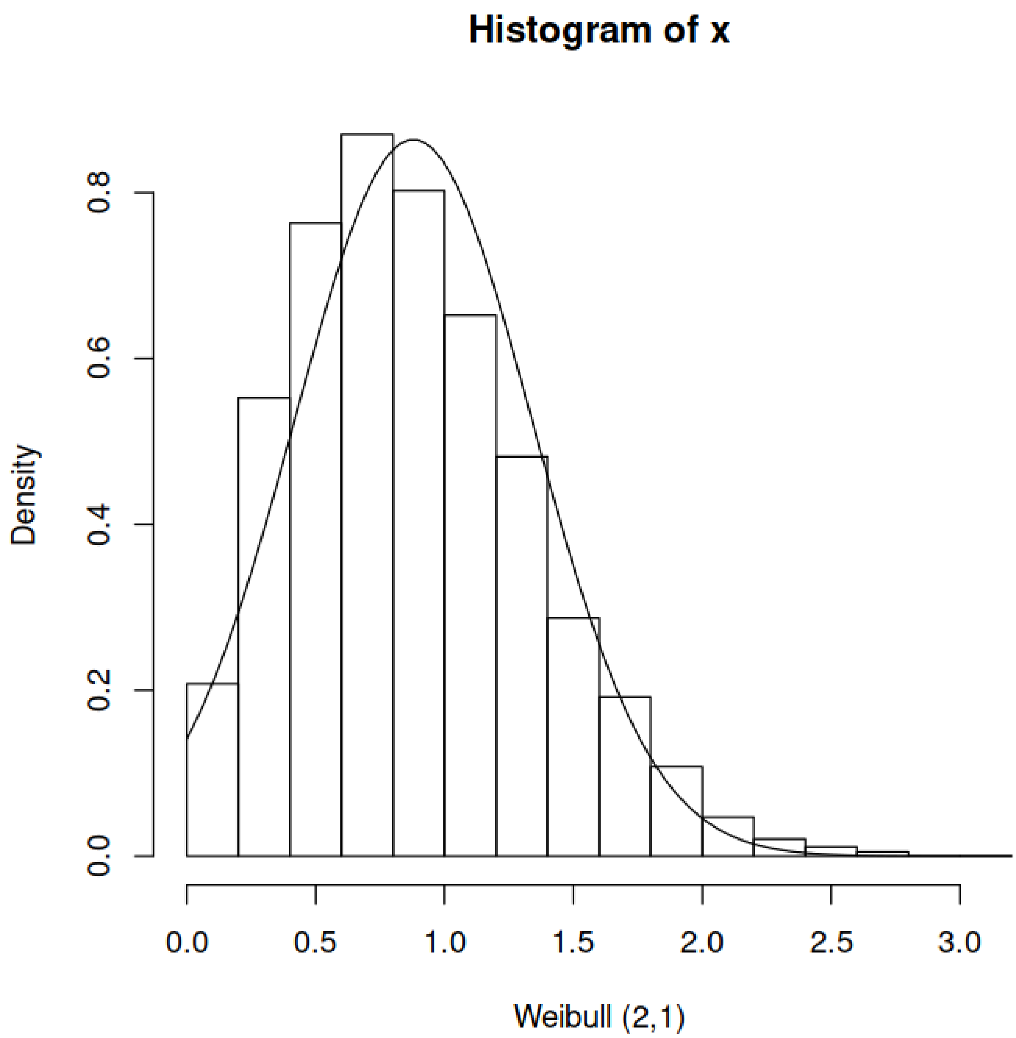



```
In [103]: plot(density(cq3), col='blue', main="estimated density")
```

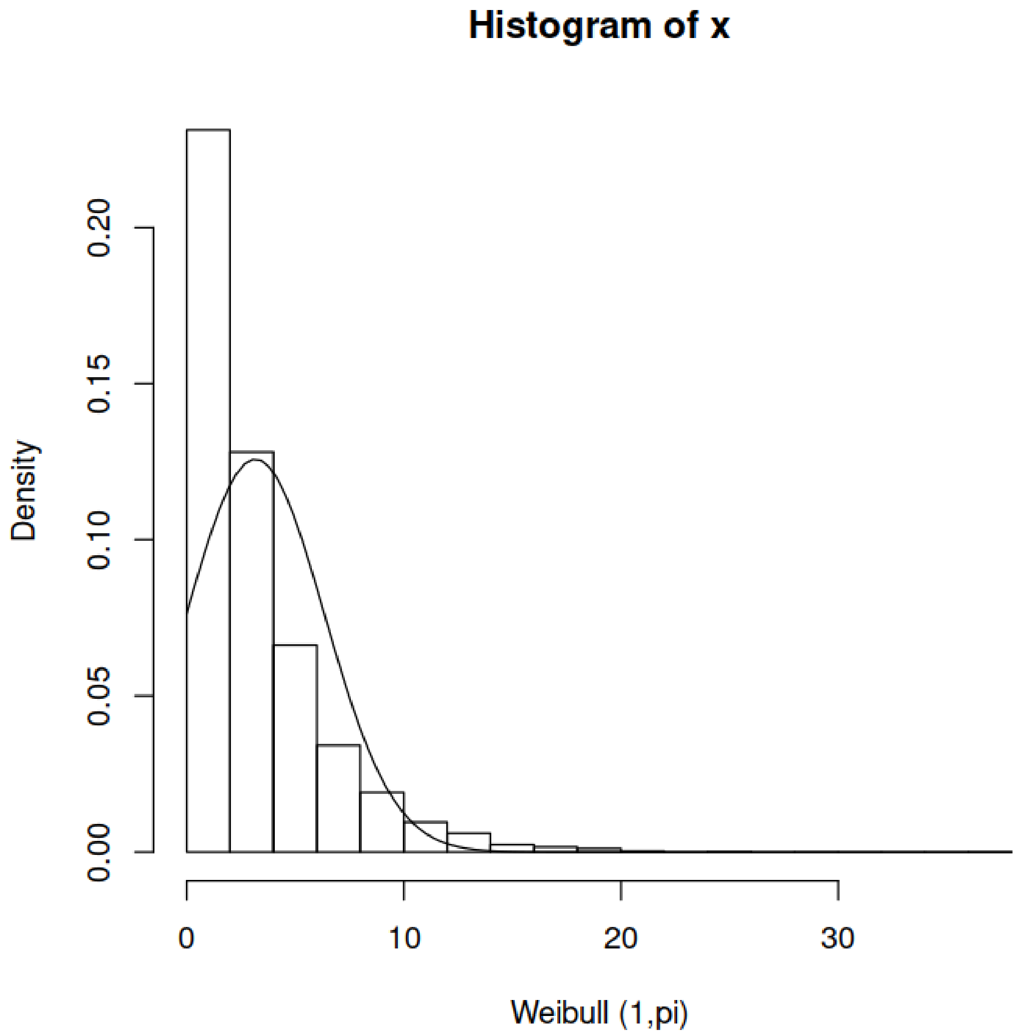


In [104]:
```
# Weibull distribution

wd1 <- rweibull(N, shape = 2, scale = 1)
wd2 <- rweibull(N, shape = 1, scale = pi)
wd3 <- rweibull(N, shape = 5, scale = 0.5)
```

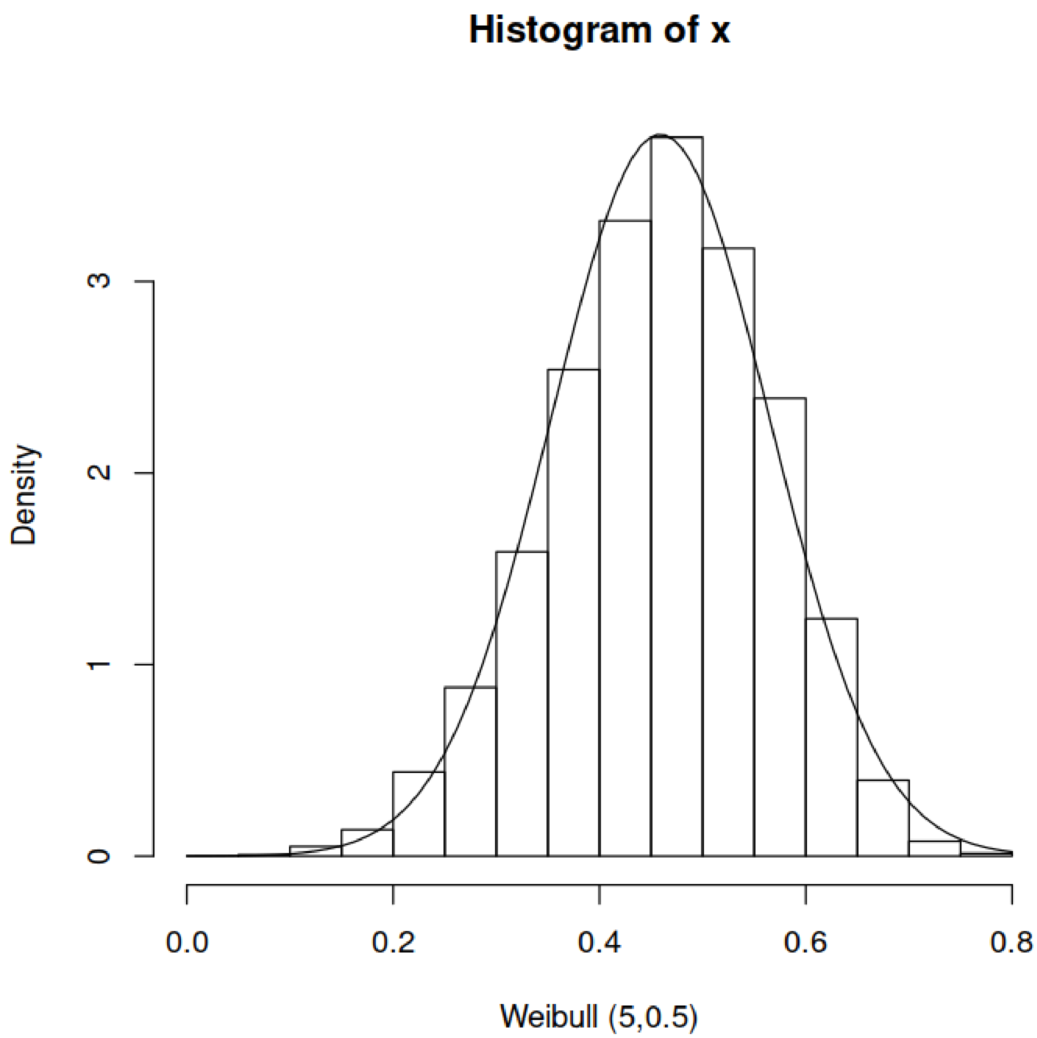
In [105]:
```
hist.with.normal (wd1, "Weibull (2,1)")
```



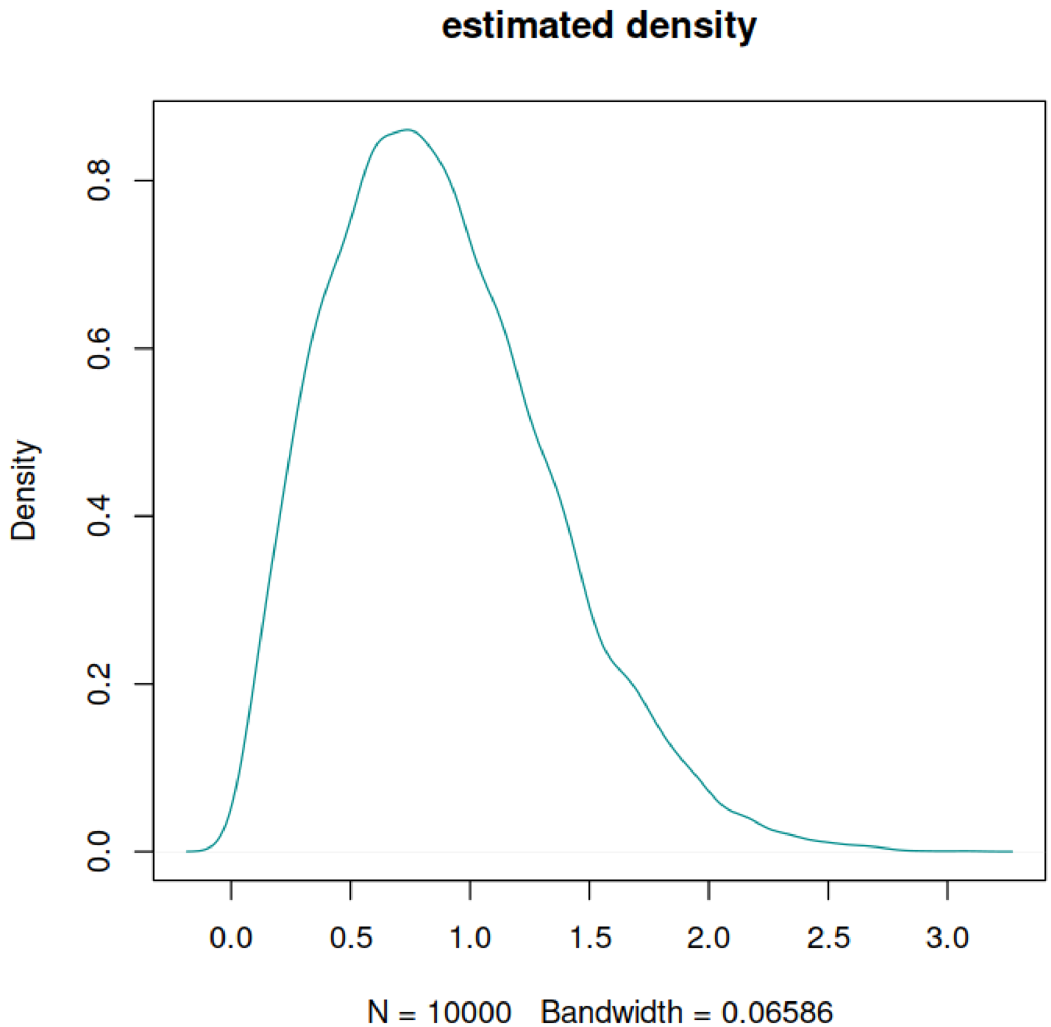

```
In [106]: hist.with.normal (wd2, "Weibull (1,pi)")
```



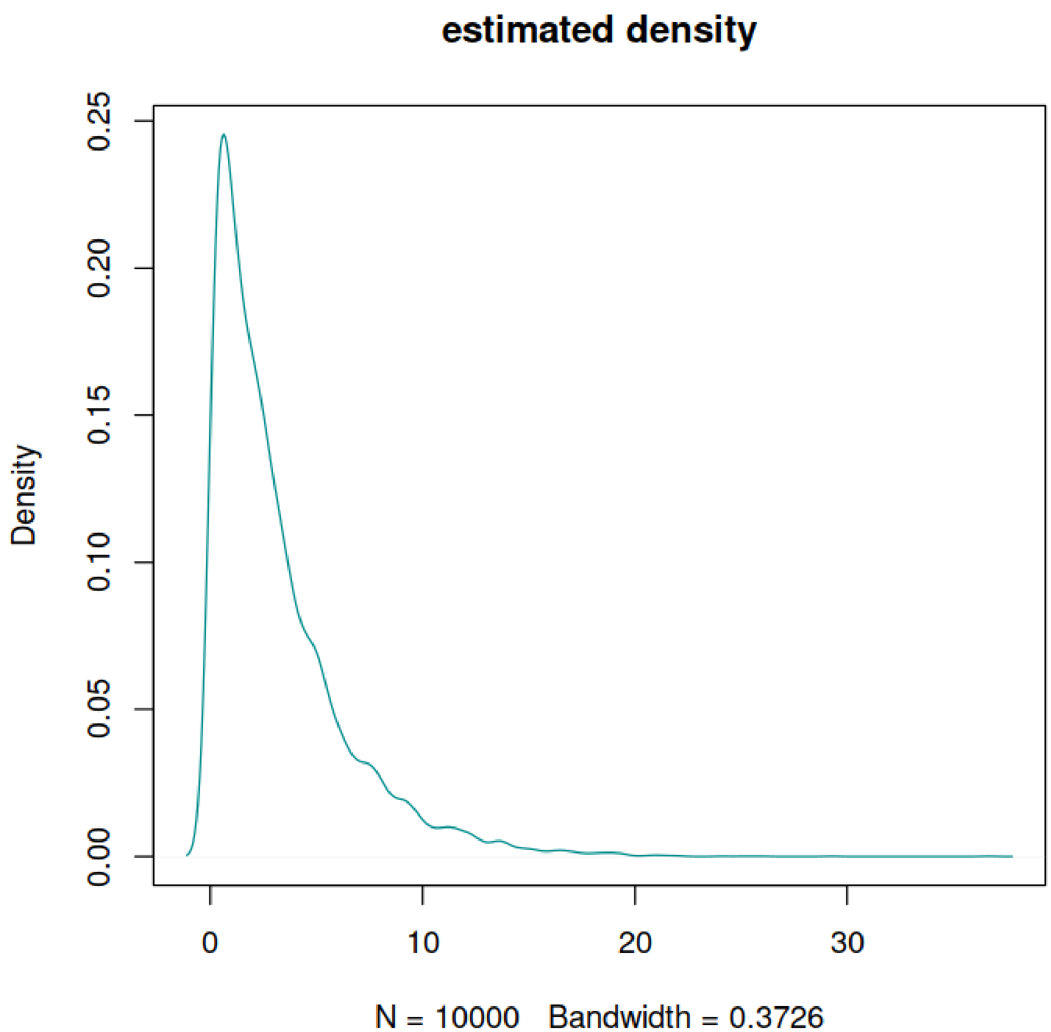

```
In [107]: hist.with.normal (wd3, "Weibull (5,0.5)")
```



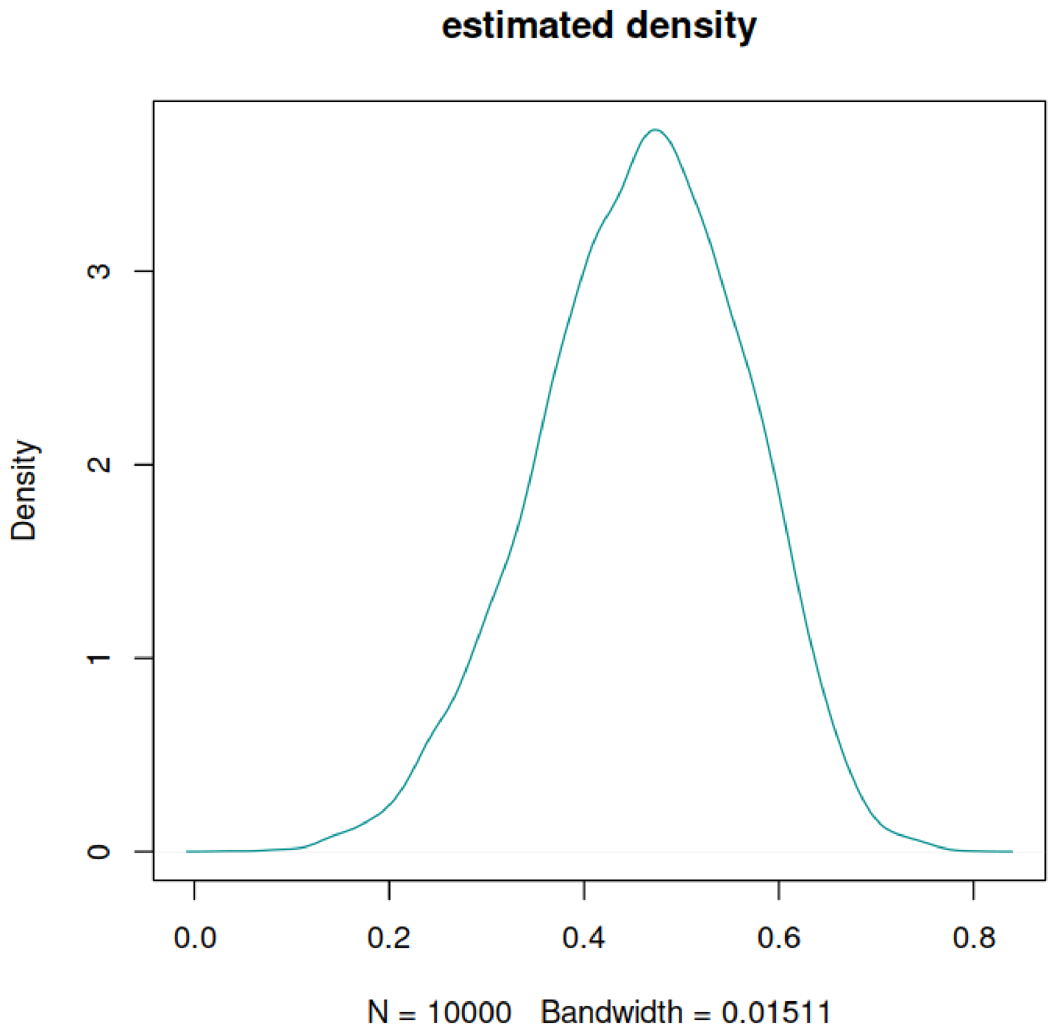

```
In [108]: plot(density(wd1), col='darkcyan', main="estimated density")
```




```
In [109]: plot(density(wd2), col='darkcyan', main="estimated density")
```




```
In [110]: plot(density(wd3), col='darkcyan', main="estimated density")
```

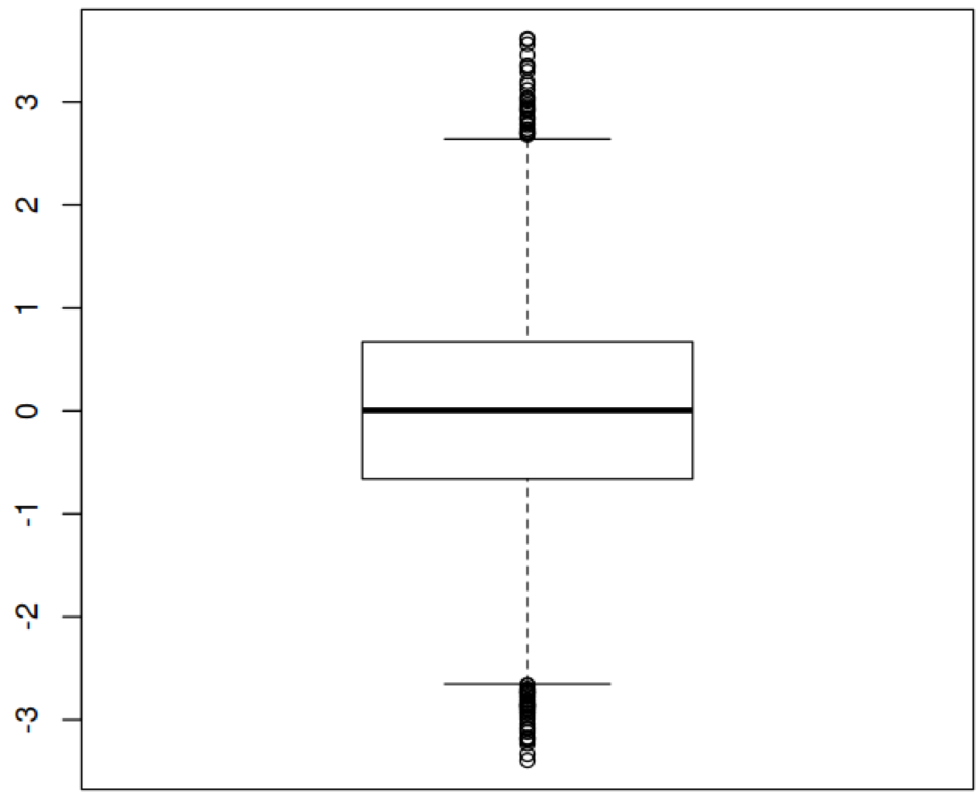


Now let's plot these samples

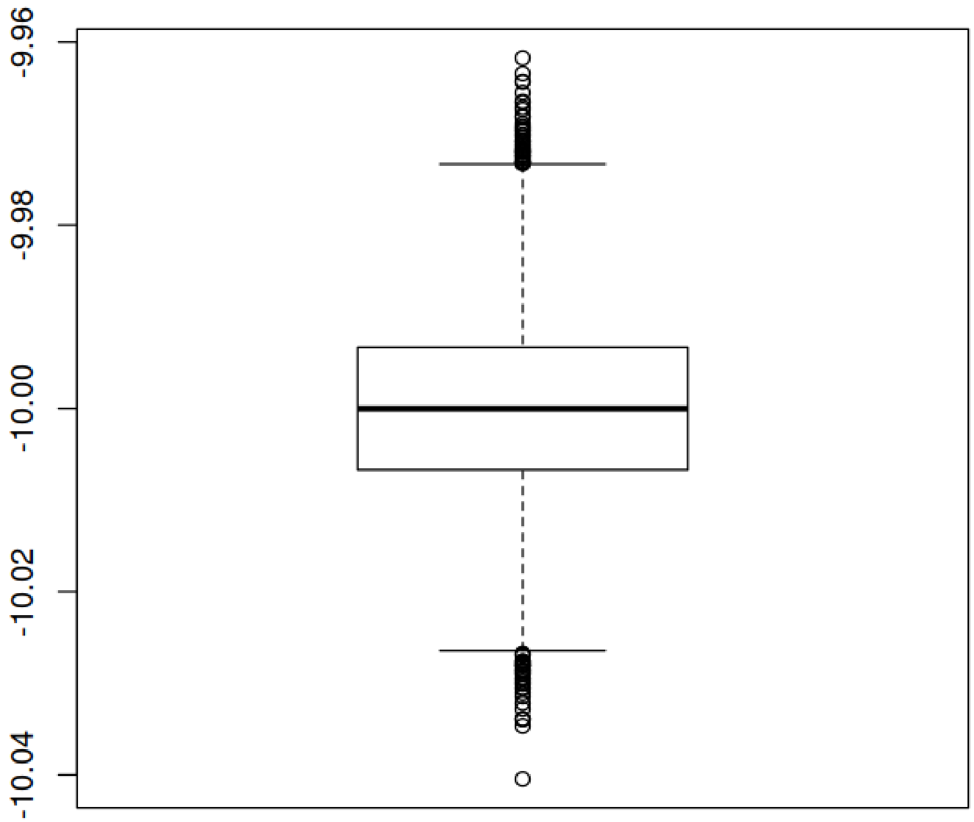
```
In [111]: library(MASS)
```

the boxplot is a very useful 1D tool

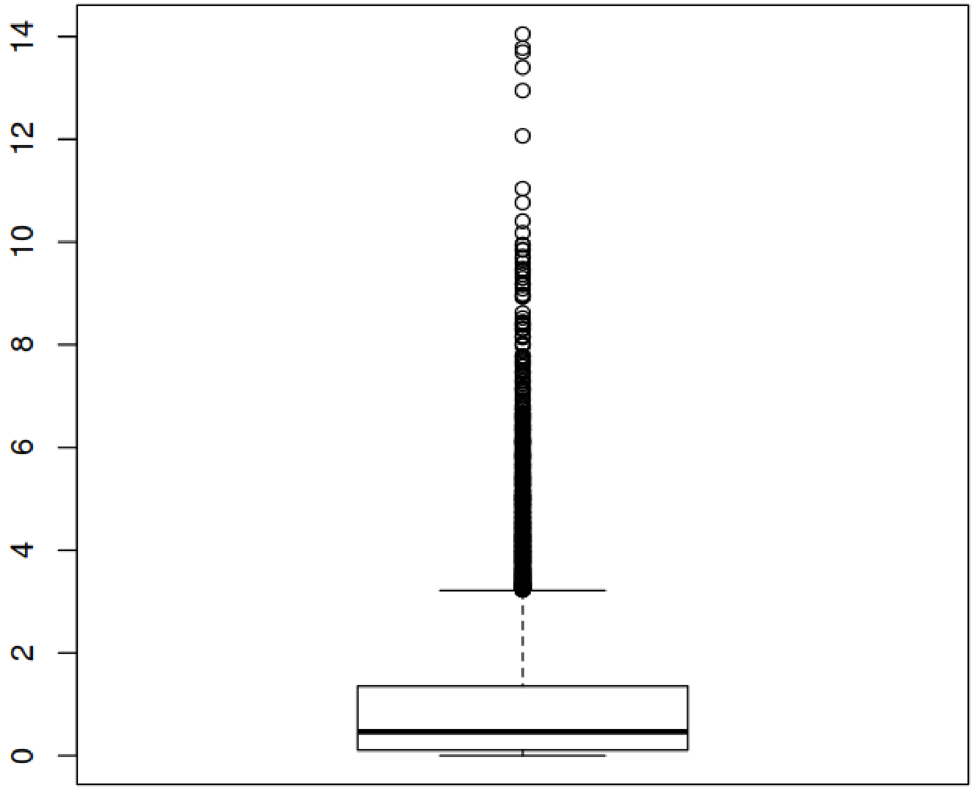
```
In [112]: boxplot(nd1)
```



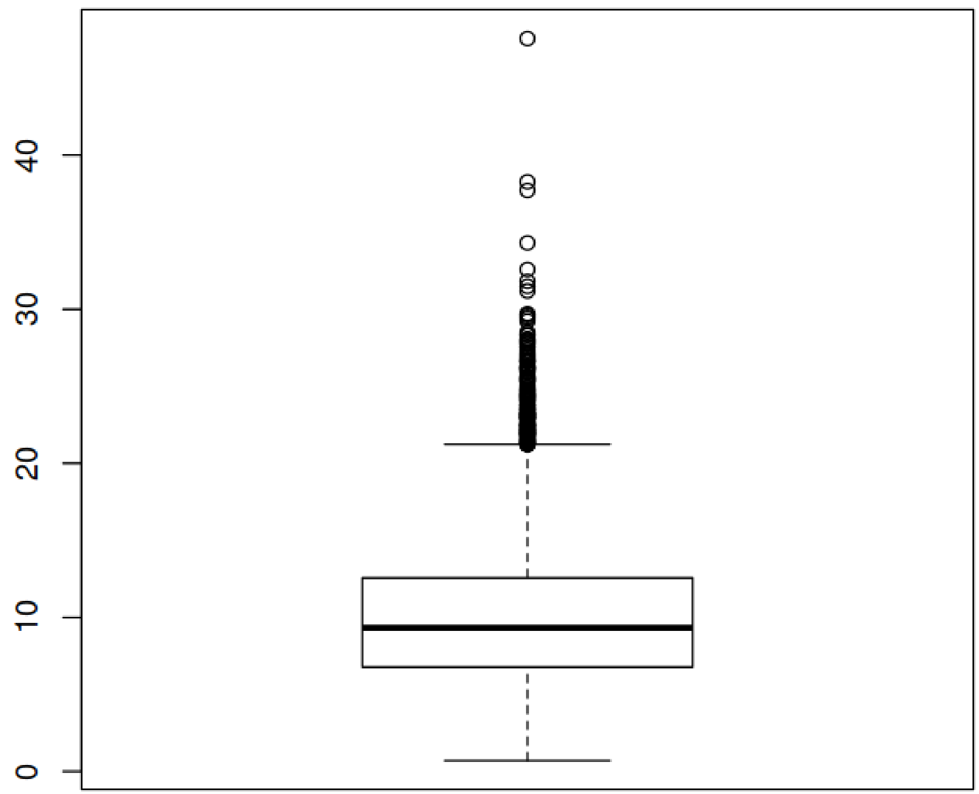

```
In [113]: boxplot(nd3)
```



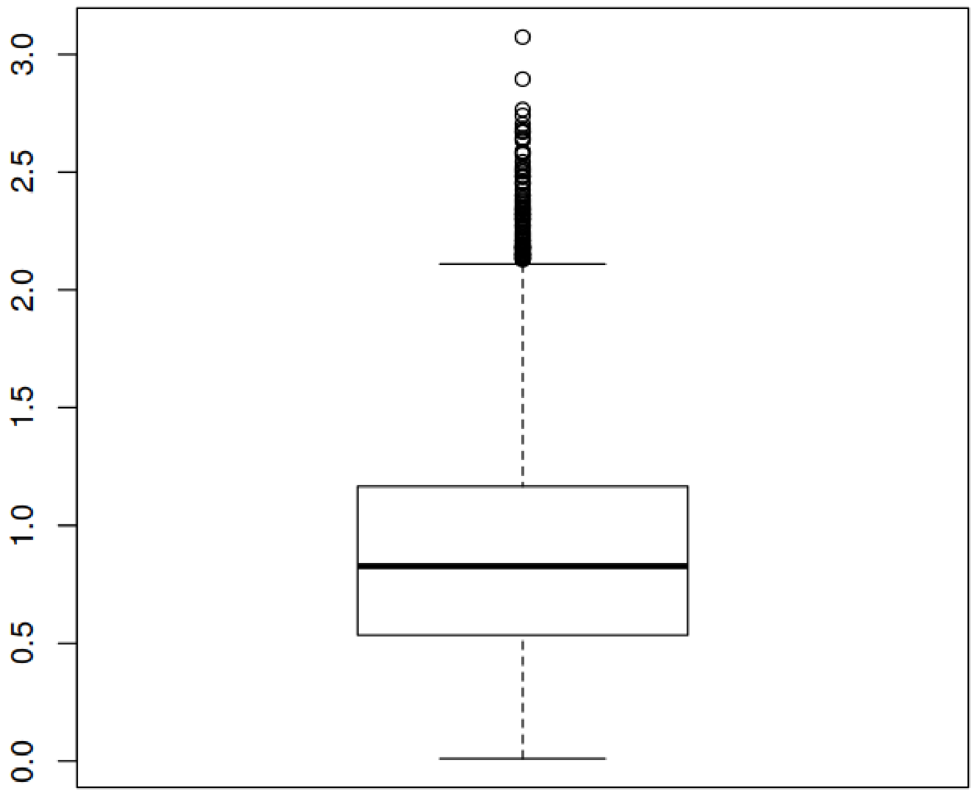

```
In [114]: boxplot(cq1)
```



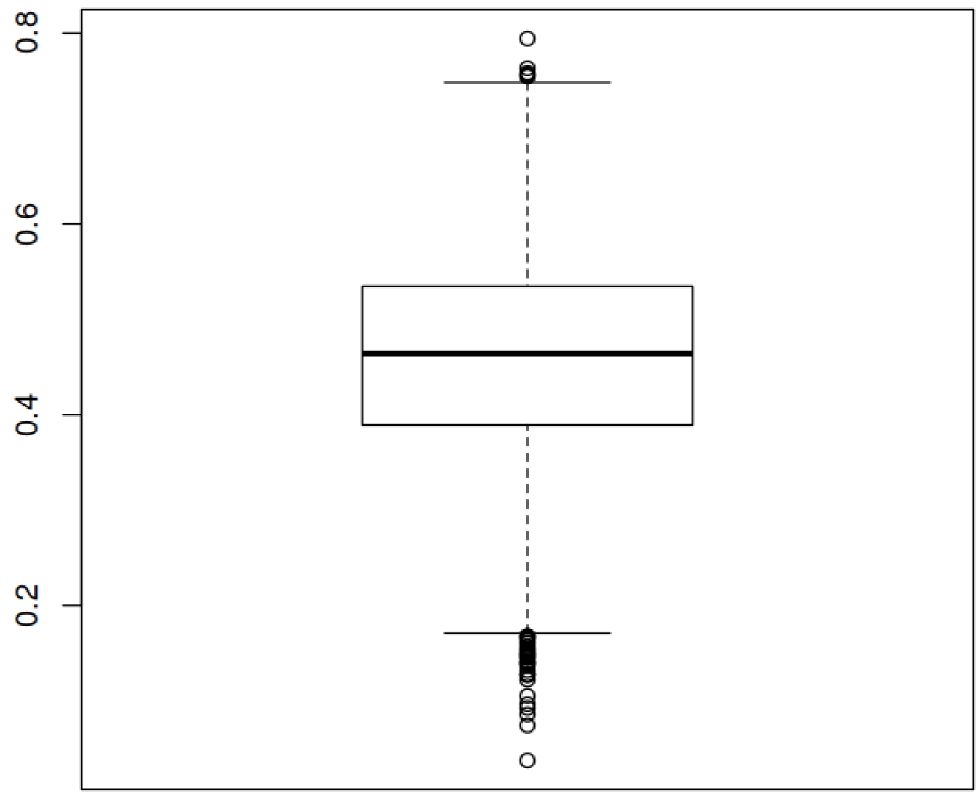

```
In [115]: boxplot(cq3)
```




```
In [116]: boxplot(wd1)
```

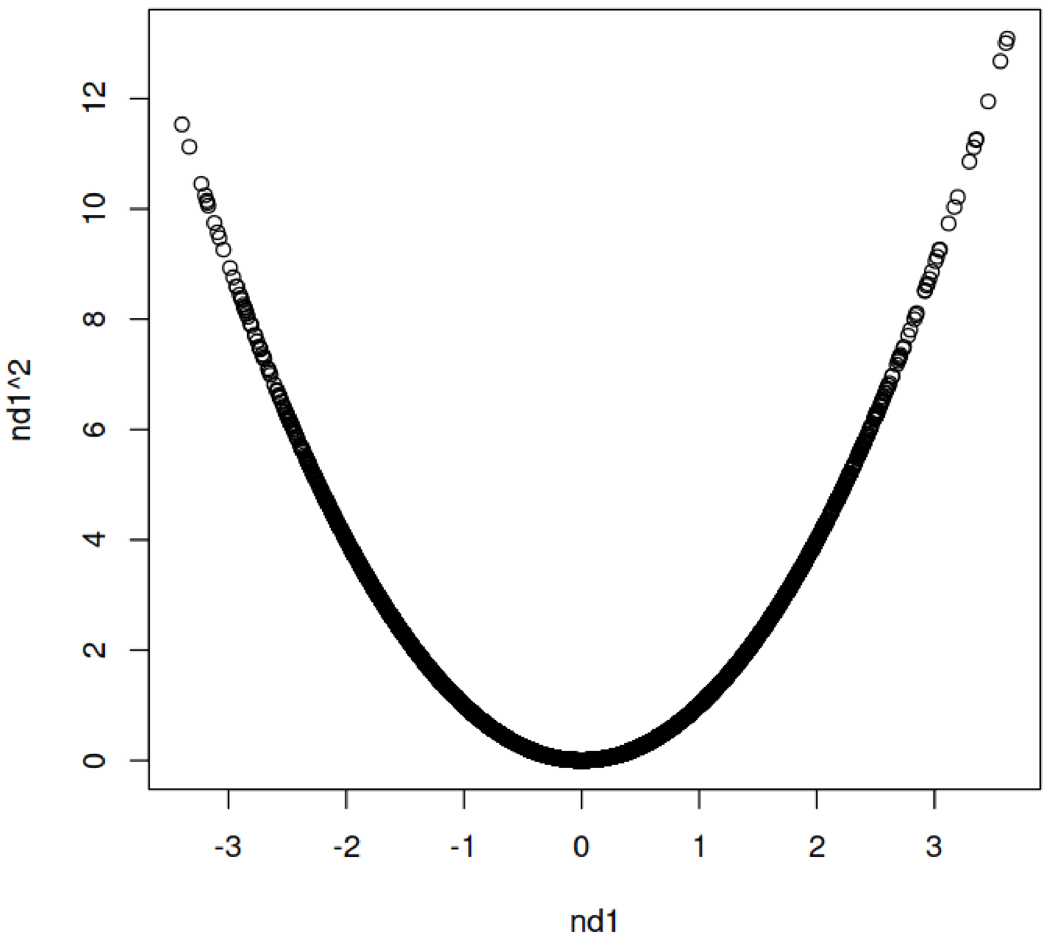


In [117]: boxplot(wd3)

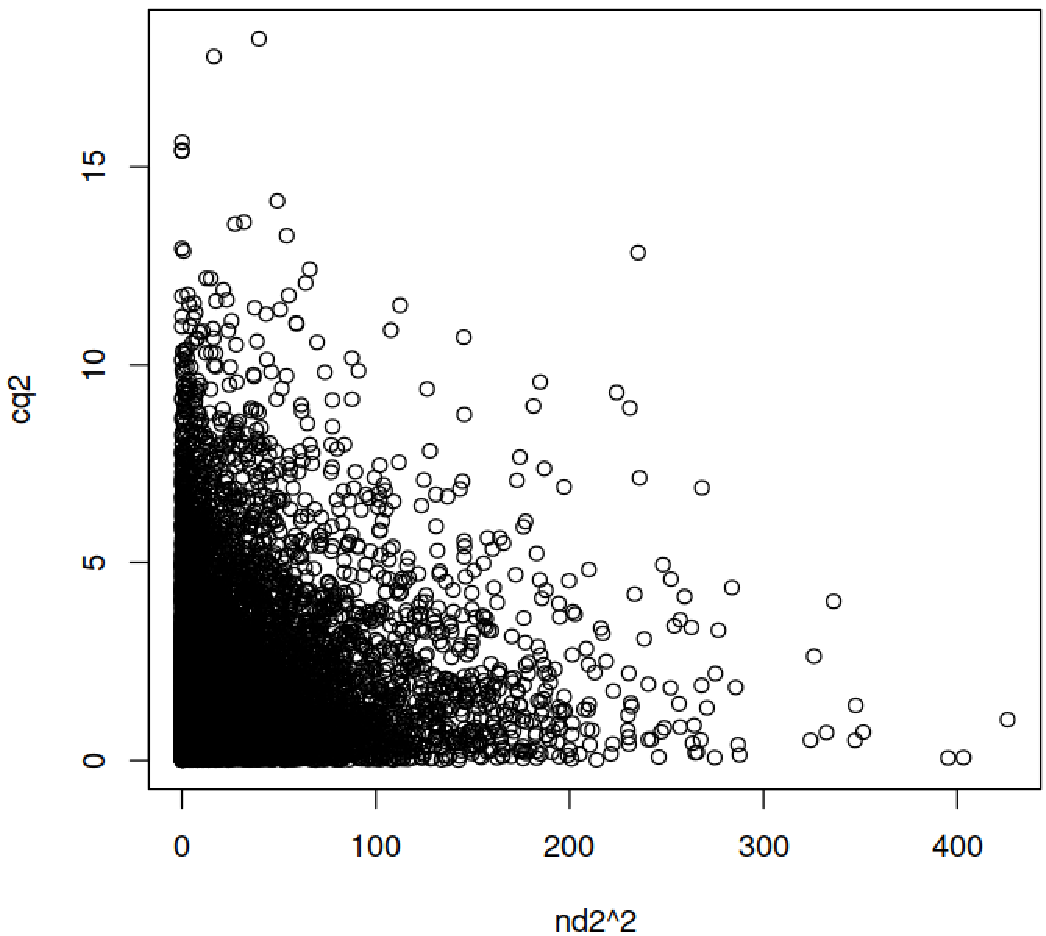


various 2D plots

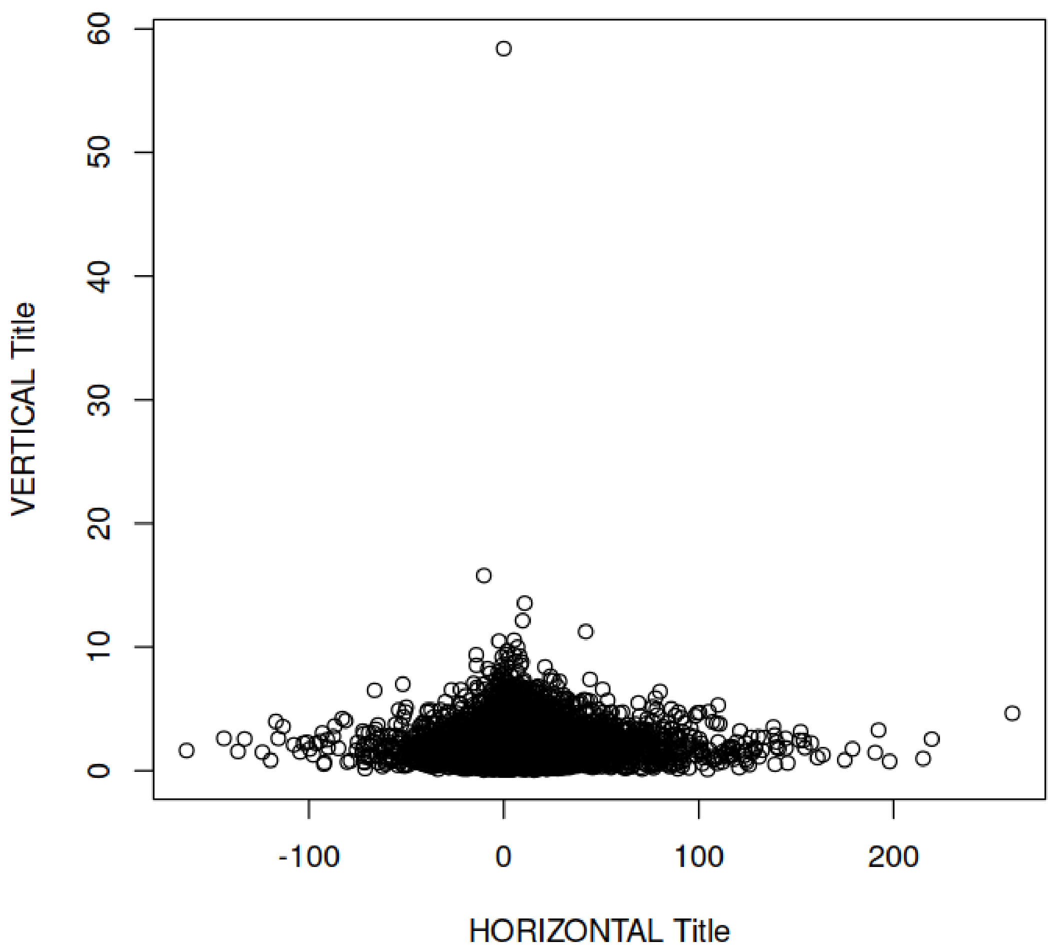
```
In [118]: plot(nd1, nd1^2)
```



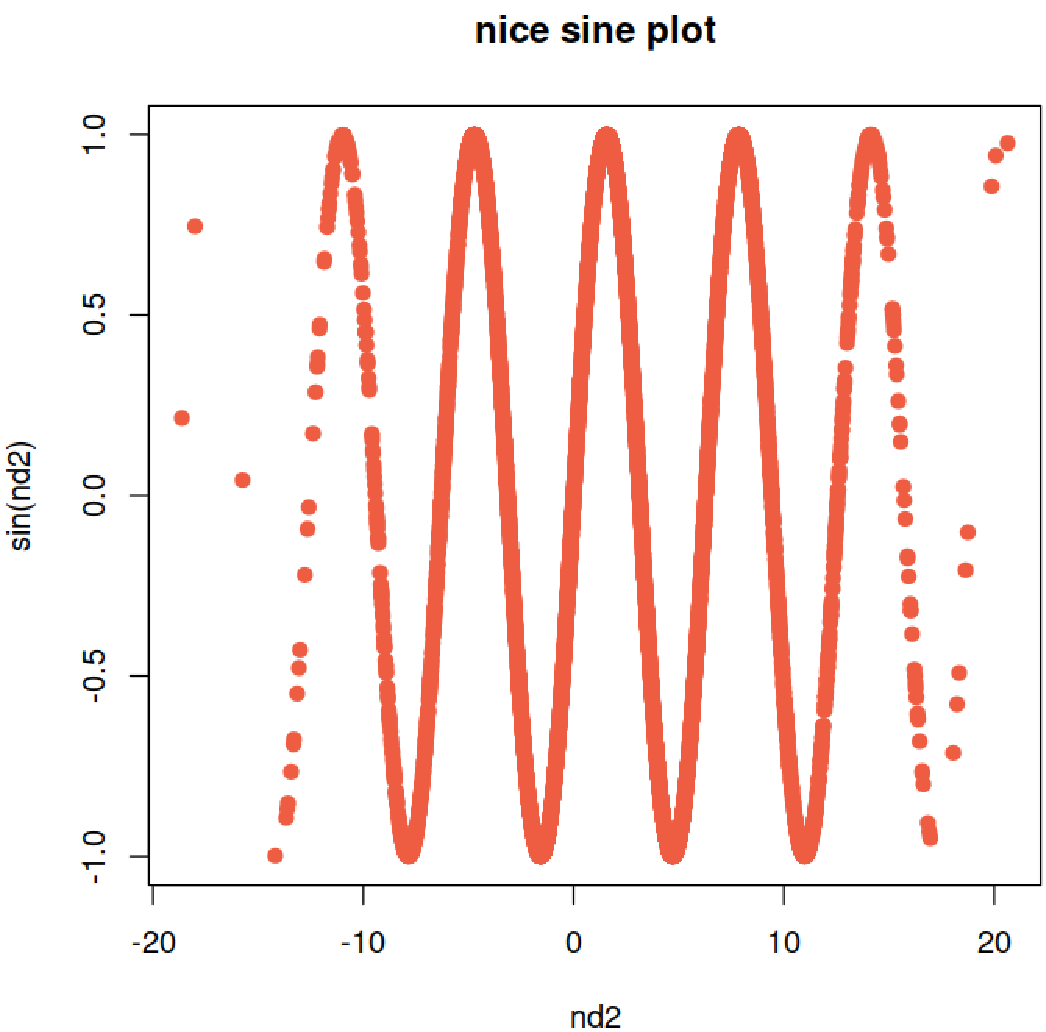

```
In [119]: plot(nd2^2, cq2)
```




```
In [120]: plot(nd2*wd2, wd1/wd3, xlab = "HORIZONTAL Title", ylab = "VERTICAL Title")
```

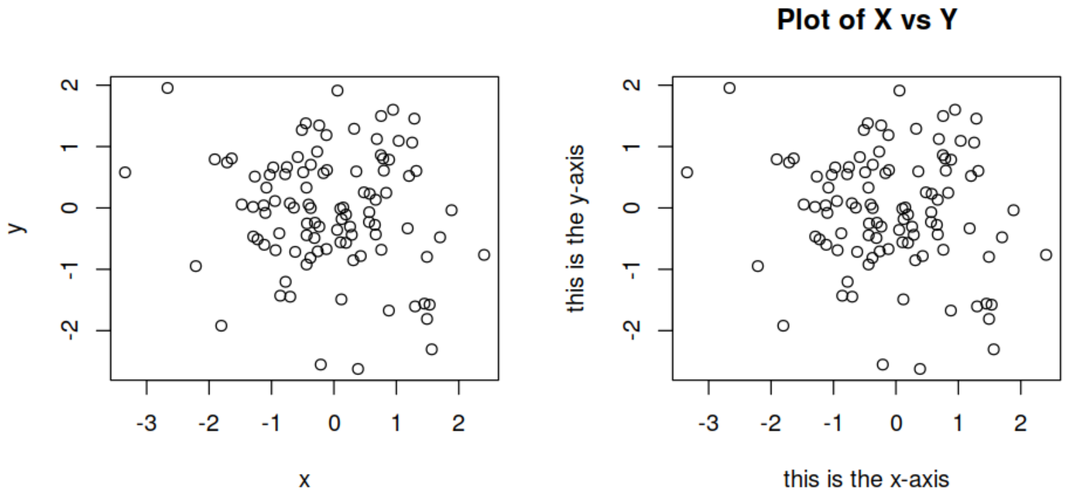



```
In [121]: plot(nd2,sin(nd2), main = "nice sine plot", pch = 19, col="tomato2")
```



1.18 Plots and graphics

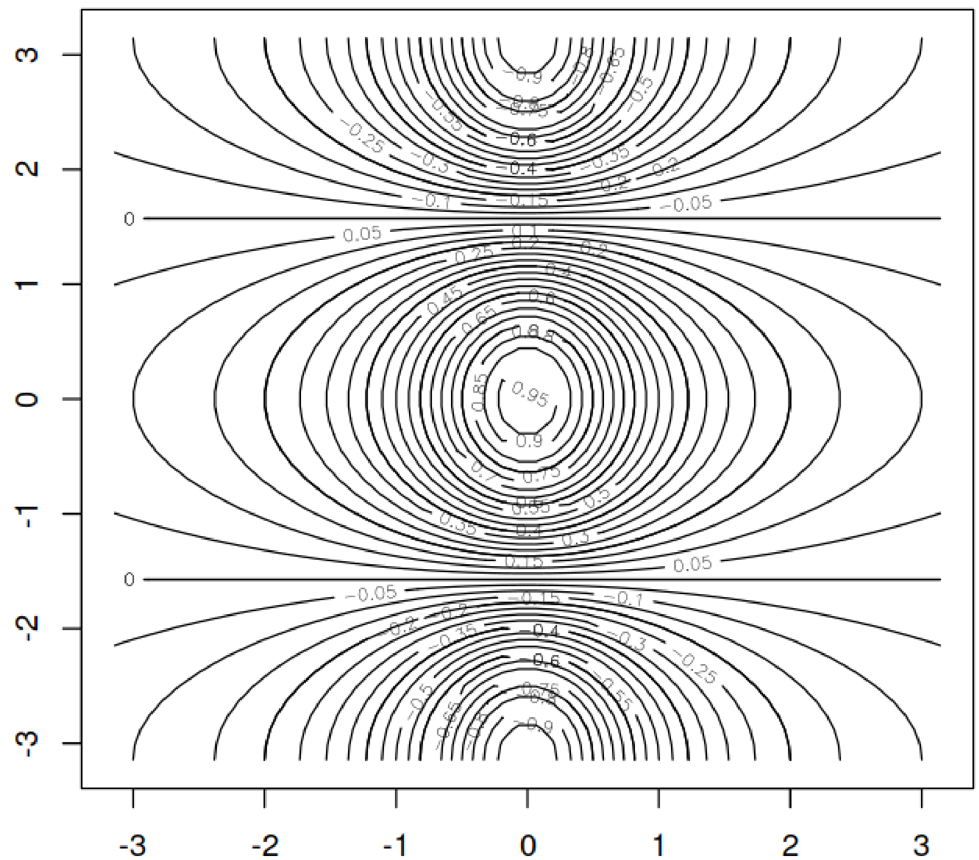
```
In [140]: x <- rnorm(100)
          y <- rnorm(100)
          options(repr.plot.width=8, repr.plot.height=4)
          par(mfrow=c(1,2))
          plot(x,y)
          plot(x,y,xlab="this is the x-axis",ylab="this is the y-axis",main="Plot of X vs Y")
```



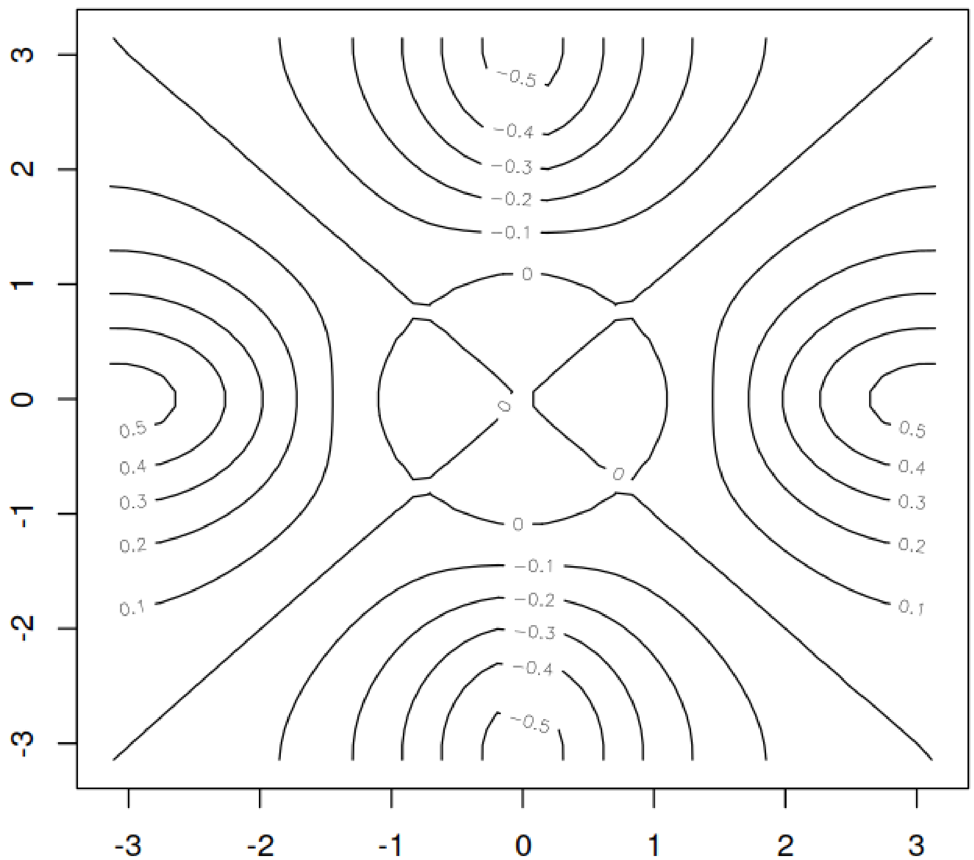

```
In [123]: pdf("Figure.pdf")

          plot(x,y,col="green")

          dev.off()
```

png: 2

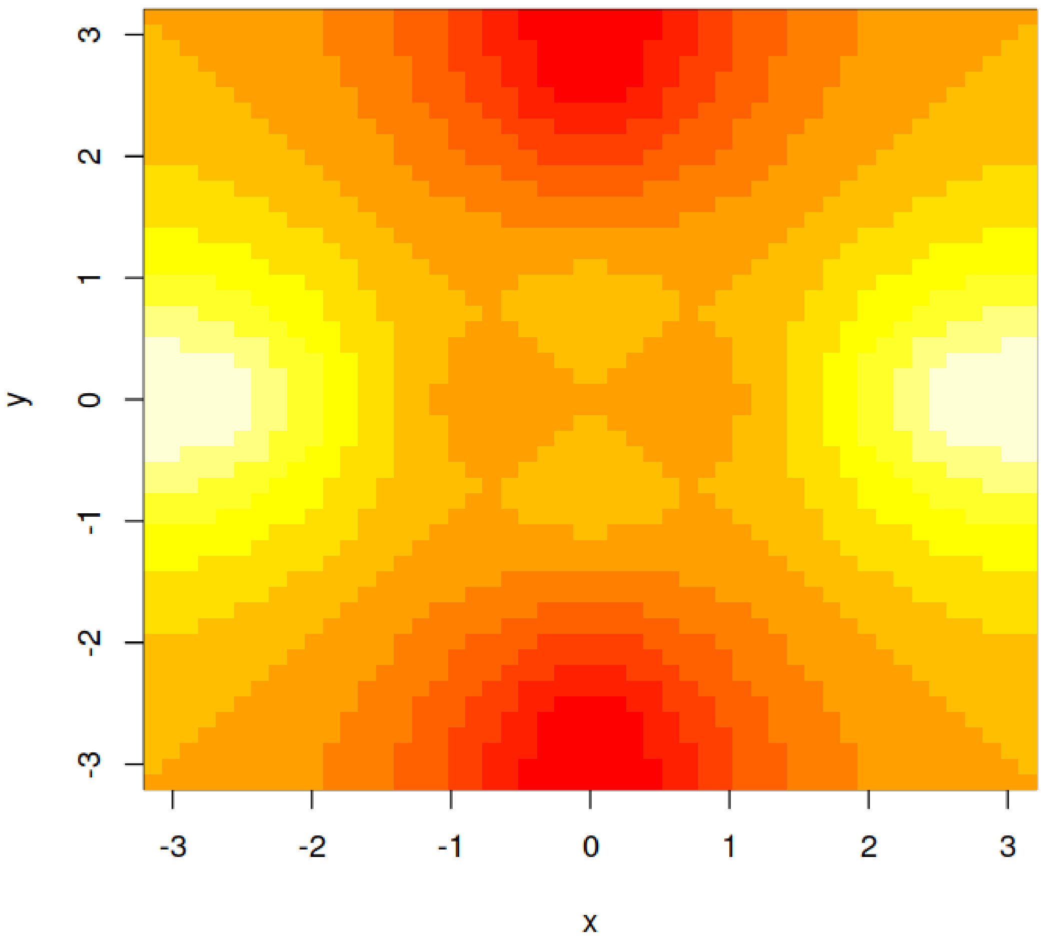
```
In [124]: x <- seq(-pi,pi,length=50)
          y <- x
          options(repr.plot.width=6, repr.plot.height=6)
          f <- outer(x,y,function(x,y)cos(y)/(1+x^2))
          contour(x,y,f)
          contour(x,y,f,nlevels=45,add=T)
```



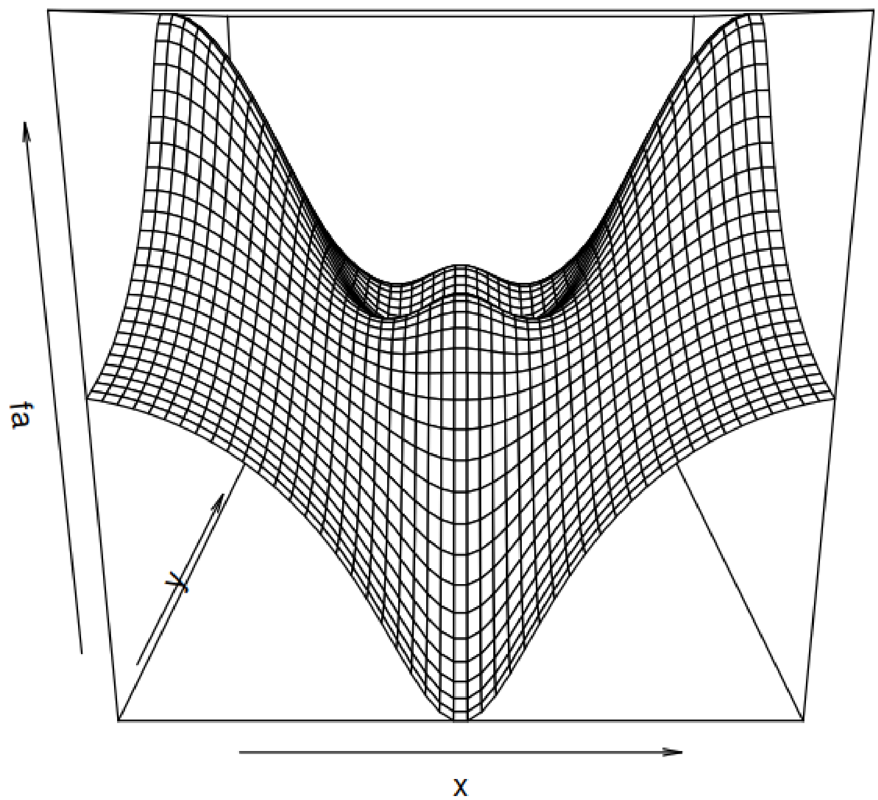
```
In [125]: fa <- (f-t(f))/2
          contour(x,y,fa,nlevels=15)
```



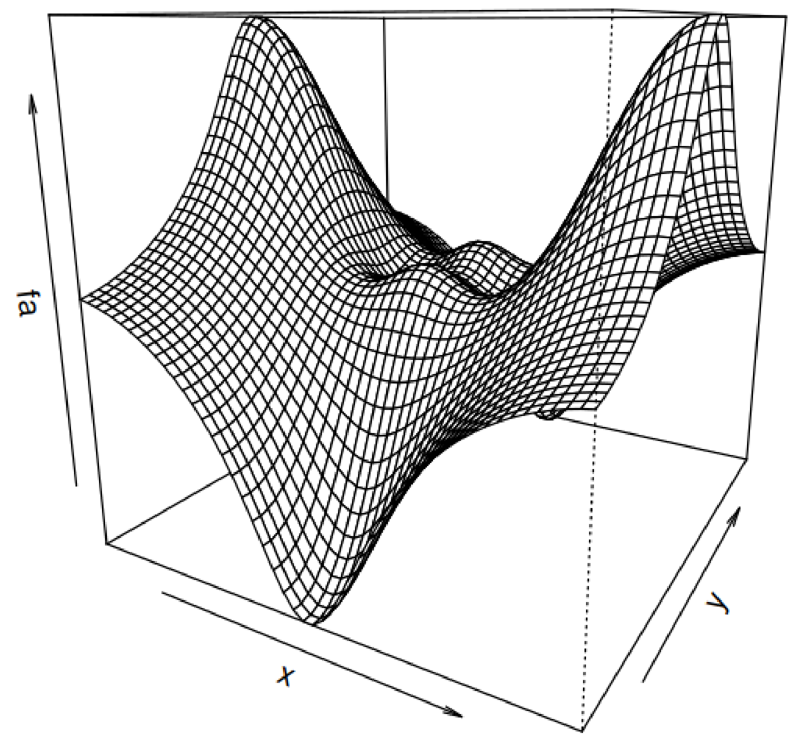

```
In [126]: image(x,y,fa)
```



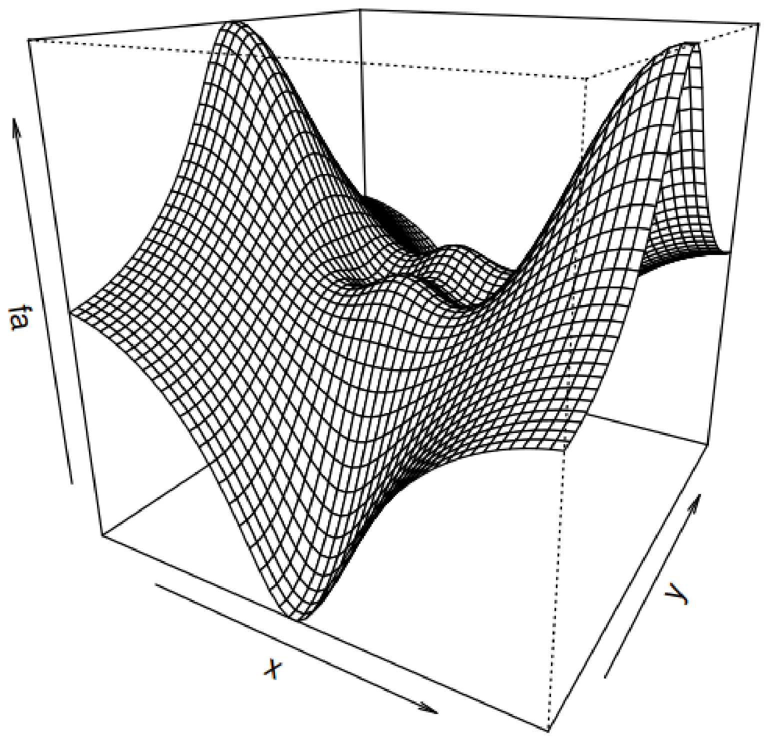
In [127]: persp(x,y,fa)



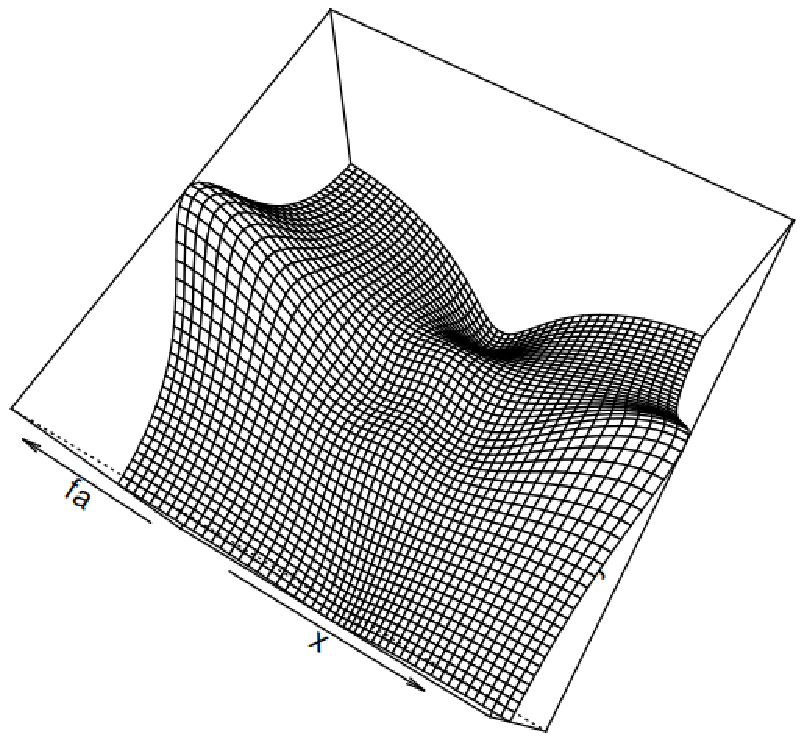

```
In [128]: persp(x,y,fa,theta=30)
```



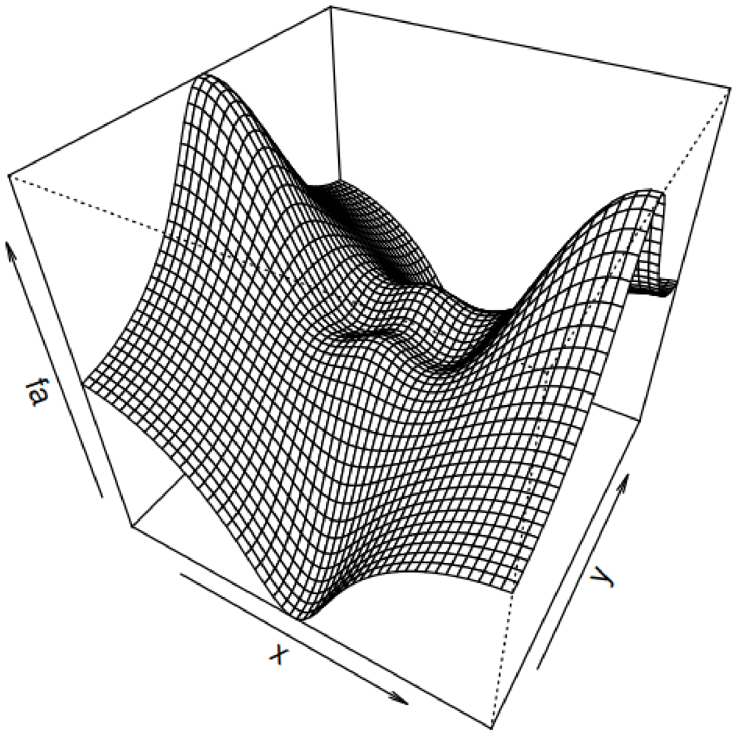

```
In [129]: persp(x,y,fa,theta=30,phi=20)
```




```
In [130]: persp(x,y,fa,theta=30,phi=70)
```

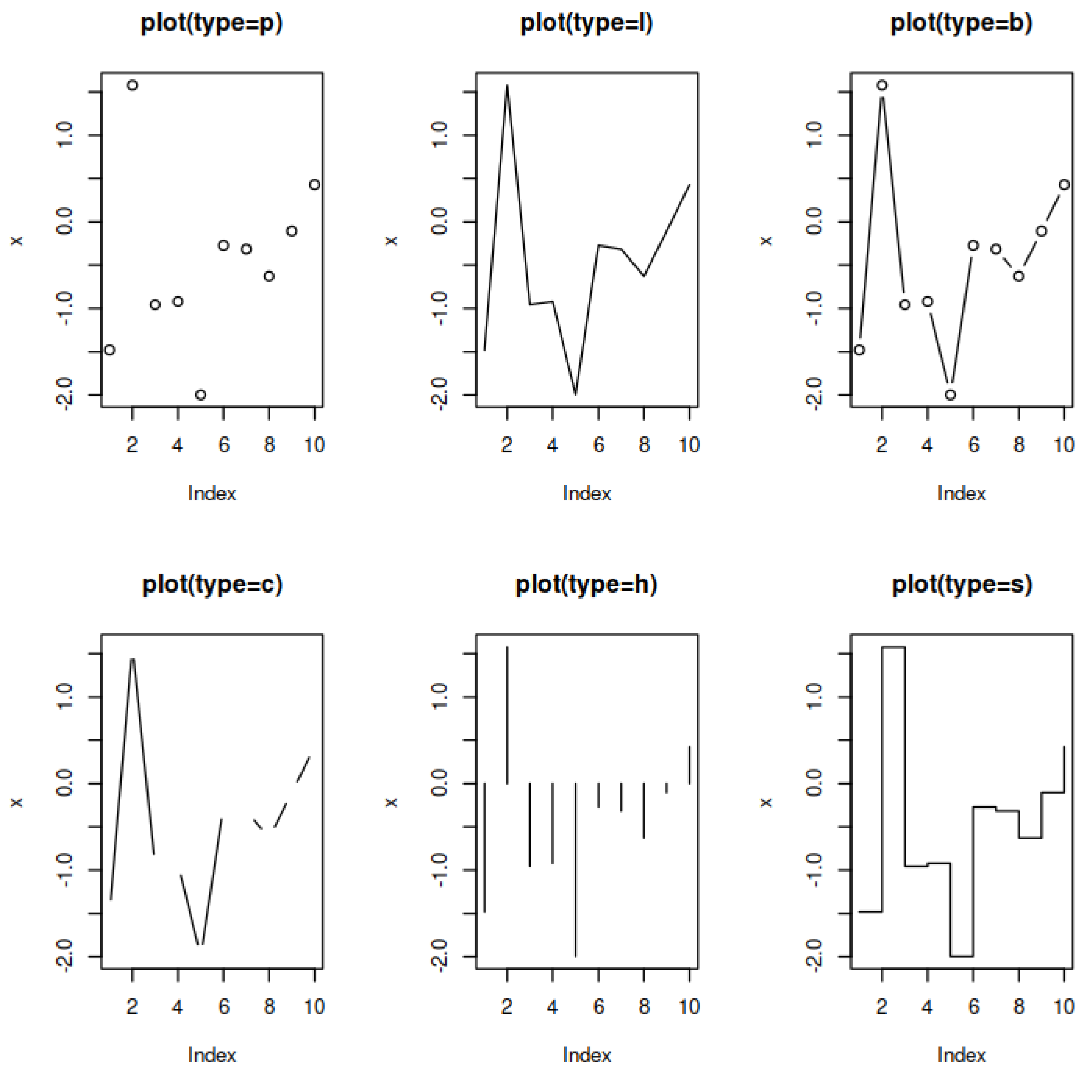



```
In [131]: persp(x,y,fa,theta=30,phi=40)
```



Still more plots ...

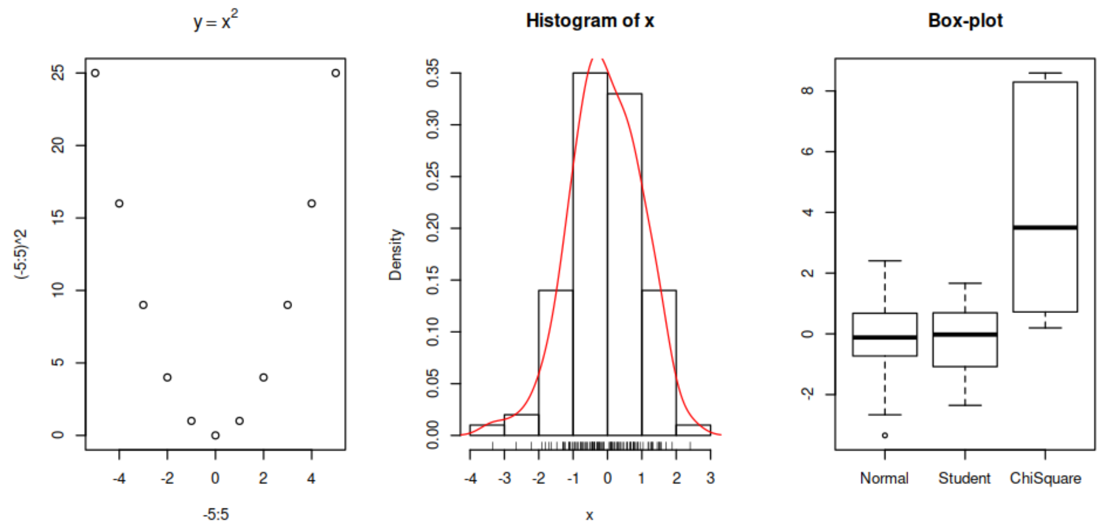
```
In [132]: par(mfrow=c(2,3)) # create a 2x3 plotting device
          set.seed(12); x <- rnorm(10)     # random draw from N(0,1)

          for (i in c("p","l","b","c","h","s")) # loop through plot types
          plot(x=x,type=i,
          main=paste("plot(type=",i,")",sep=""))
```



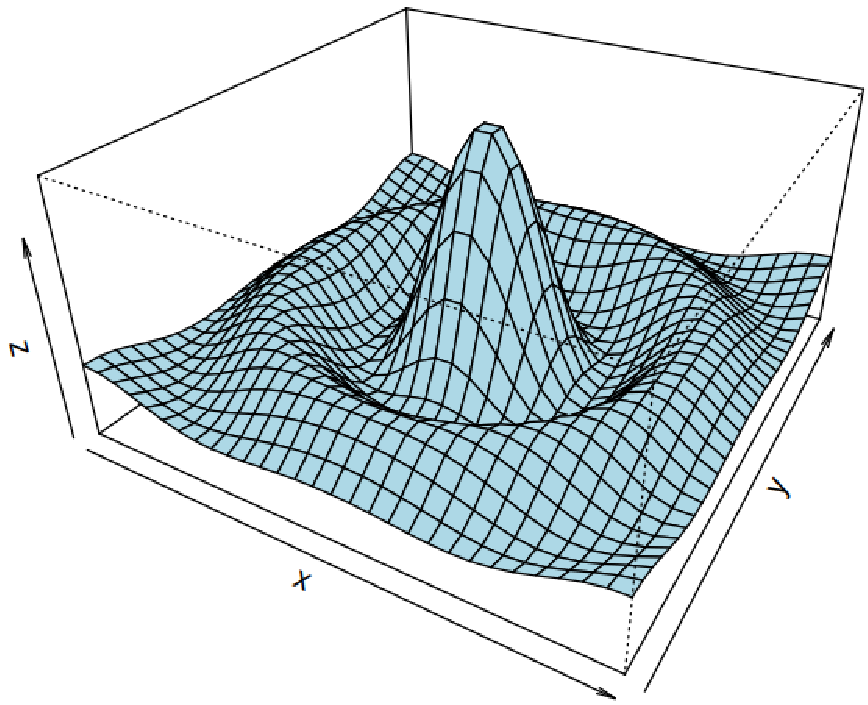
Simple 2D plots: hist(), boxplot()

```
In [141]: options(repr.plot.width=8, repr.plot.height=4)
          par(mfrow=c(1,3))        # parametrize a 1x3 plotting device

          plot(x=-5:5,y=(-5:5)^2, # simple scatter plot of y=x^2
          main=expression(y==x^2))  # with math expression in the title

          hist(x,freq=FALSE)          # histogram with prob. densities
          lines(density(x),col="red") # add red kernel density to histogram
          rug(x)                      # add rug representation to histogram

          boxplot(x,rt(10,3),rchisq(10,3), # boxplot
```

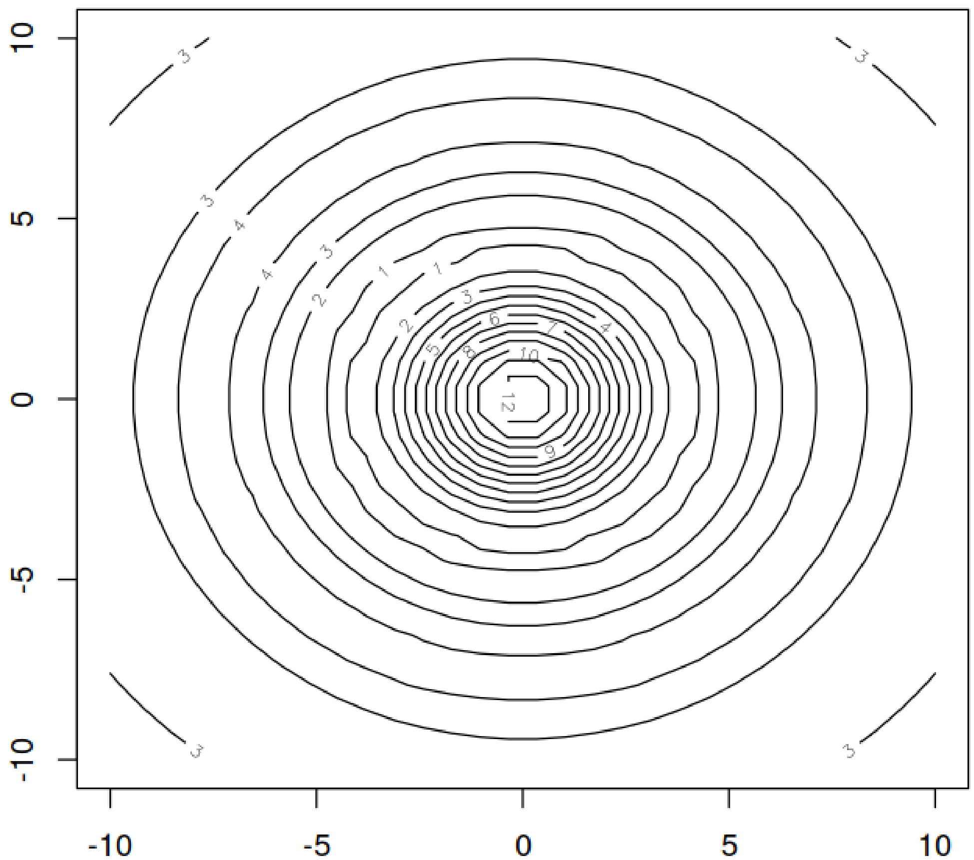
```
names=c("Normal","Student","ChiSquare"))  # series labels
title("Box-plot")                          # add title to boxplot

graphics.off()      # reset/close all graphical devices
```



In [134]:
```
## 3D plots: persp(), contour()

x <- seq(-10,10, length=30); y <- x        # x-y dimensions
f <- function(x, y) {r <- sqrt(x^2+y^2);  # z dimension
10 * sin(r)/r + 3}
z <- outer(x, y, f)    # apply f to "outer product" of x and y

persp(x, y, z, zlim=range(0,max(z)),       # 3D plot
theta=30, phi=30, expand=0.5, col="lightblue")
```




```
In [135]: contour(x, y, z) # contour plot
```



1.19 Computations with big datasets

```
In [136]: M <- matrix(rnorm(5e6),ncol=50)
          dim(M)
```

1. 100000 2. 50

Times are: user cpu, system cpu, elapsed

```
In [137]: system.time(M+1)
```

```
   user  system elapsed 
  0.006   0.016   0.024 
```

see the difference!

```
In [138]: M.df <- data.frame(M)
          system.time(M.df+1)
```

```
   user  system elapsed 
  0.014   0.017   0.032 
```

see the difference!

```
In [139]: M <- matrix(rnorm(2e6), nrow=2000)
          dim(M)

          system.time(apply(M,1,sum))              # row sums
          system.time(M %*% rep(1,1000))            # row sums
          system.time(rowSums(M))                  # row sums
```

1. 2000 2. 1000

```
   user  system elapsed 
  0.038   0.005   0.043 
```

```
   user  system elapsed 
  0.005   0.000   0.005 
```

```
   user  system elapsed 
  0.006   0.000   0.006 
```