```
####################################################################################################
##Introductory R script
####################################################################################################

## By Marius Hofert for the book "Quantitative Risk Management: Concepts, Techniques and Tools" by Alexander J. McNeil, R�diger Frey and Paul Embrechts, published by Princeton University Press in 2015 (revised 2nd edition, 1st edition 2005).
## http://www.math.uwaterloo.ca/~mhofert/
## http://www.qrmtutorial.org/

### Comments ###################################################################

## Q: What is R?
## A: - R is a *free* software environment for *statistical* computing and *graphics*
##    - R was created by *R*obert Gentleman and *R*oss Ihaka in 1993.
##    - Since mid-1997, R is developed by the *R Development Core Team* (and
##      contributors)

## Q: Why R?
## A: - *Packages* (both the available ones and the possibility to write
##      your own)
##    - Ability to write *readable* code (focus is on main aspects of a problem)
##    - High(er) level (optimization, run time, debugging, parallel computing etc.)
##

## Q: Where to find R or help on R?
## A: - Main website: https://www.r-project.org/
##    - CRAN: https://cran.r-project.org/
##      + Packages (check 'Published', 'Reference manual', 'Vignettes', 'Package source')
##      + Task Views (check Finance -> rugarch or Multivariate -> copula)
##      + Manuals ("An Introduction to R" (detailed basics) and
##                 "Writing R Extensions" (package development))
##      + FAQ, in particular, FAQ 2.7 on http://cran.r-project.org/doc/FAQ/R-FAQ.html#What-documentation-exists-for-R_003f
##    - Externally (outside R or CRAN):
##      + Google ('r-help') or R-related search engines;
##        see http://tolstoy.newcastle.edu.au/R/ and http://finzi.psych.upenn.edu/
##      + R Help and other mailing lists; see https://stat.ethz.ch/mailman/listinfo/r-help
##      + Stackoverflow (tag 'R'); see http://stackoverflow.com/questions/tagged/r
##        => provide a minimal working example, see https://en. wikipedia.org/wiki/Minimal_Working_Example
##      + R graph gallery; see http://www.r-graph-gallery.com/
##      + R blog; see http://www.r-bloggers.com/
##    - Internally (from within R):
##      + Browser-based help: help.start() -> Search Engine & Keywords
##      + '?' (e.g., ?uniroot) or 'help("[[")' (for specific functions)
##        => Study the examples on the help files!
##      + Study the source code (for more ``hidden'' functions, see pp. 43 in
##        http://cran.r-project.org/doc/Rnews/Rnews_2006-4.pdf)

## Q: How can I work with R?
## A: - Software interfaces:
##      + RStudio (recommended): http://www.rstudio.com/
##      + Emacs + ESS
##    - Workflow:
##      + Write an R script (.R file) containing the source code.
##      + Execute it line-by-line (paragraph-by-paragraph etc.) or the whole script
##        at once (if in batch mode, e.g., on a computer cluster).

## Q: How to install (the latest version of) a package?
## A: - From CRAN (release):
##      install.packages("mypackage")
##    - From R-Forge (snapshot):
##      install.packages("mypackage", repos = "http://R-Forge.R-project.org")

## Q: What to watch out for when programming?
## A: - Theoretical challenges (e.g., curse of dimensionality, e.g., for computing
##      P(a < X <= b) in high dimensions)
##    - Design errors (code correct, but model wrong)
##    - Programming language related issues (R vs C vs Fortran; if you need
##      to implement your own root-finding procedure, errors are more likely)
##    - Syntactic errors (code does not run; typically easy to detect; useful tools:
##      traceback(), debug() or browser())
##    - Semantic errors (code on its own correct, but does not what was intended;
##      test your code, use plots!)
##    - Numerical errors (often undetected unless code is properly tested)
##    - Warnings (are useful! For example, if the optimum in an optimization
##      procedure has not yet been reached)
##    - Measuring run time (system vs wall clock; depends
```

```
##      on architecture, programming style, compiler, current workload, etc.; use
##      benchmarks; can also be useful for detecting semantic errors)
##    - Scaling (bigger simulations; if possible, use parallel's mclappy() and
##      parLapply() for multi-core and multi-node computations)

## Note: The code below is a medley of Appendix A of the manual
##       "An Introduction to R" on http://cran.r-project.org/manuals.html


### Simple manipulations; numbers and vectors #################################

## Simple manipulations
1+2
1/2
1/0 # in R, Inf and -Inf exist and R can often deal with them correctly
0/0 # ... also NaN = 'not a number' is available; 0/0, 0*Inf, Inf-Inf lead to NaN
x <- 0/0 # store the result in 'x'
class(x) # the class/type of 'x'; => NaN is still of mode 'numeric'
class(Inf) # Inf is of mode 'numeric' (although mathematically not a number); helpful in optimizations

## Vectors (data structure which contains objects of the same mode)
numeric(0) # the empty numeric vector
length(numeric(0)) # its length
x <- c(1, 2, 3, 4, 5) # numeric vector
x # print method
(y <- 1:5) # another way of creating such a vector (and *printing* the output via '()')
(z <- seq_len(5)) # and another one (see below for the 'why')
z[6] <- 6 # append to a vector (better than z <- c(z, 6)); (much) more comfortable than in C/C++
z

## Note: We can check whether the R objects are the same
x == y # component wise numerically equal
identical(x, y) # identical as objects? why not?
class(x) # => x is a *numeric* vector
class(y) # => y is an *integer* vector
all.equal(x, y) # numerical equality; see argument 'tolerance'
identical(x, as.numeric(y)) # => also fine

## Numerically not exactly the same
x <- var(1:4)
y <- sd(1:4)^2
all.equal(x, y) # numerical equality
x == y # ... but not exactly
x-y # numerically not 0
## See also https://cran.r-project.org/doc/FAQ/R-FAQ.html#Why-doesn_0027t-R-think-these-numbers-are-equal_003f

## Watch out
n <- 0
1:n # not the empty sequence but c(1, 0); caution in 'for loops': for(i in 1:n) ...!
seq_len(n) # better: => empty sequence
seq_along(c(3, 4, 2)) # 1:3; helpful to 'go along' objects

## Watch out
1:3-1 # ':' has higher priority; note also: the '-1' is recycled to the length of 1:3
1:(3-1)

## Some functions
(x <- c(3, 4, 2))
length(x) # as seen above
rev(x) # change order
sort(x) # sort in increasing order
sort(x, decreasing = TRUE) # sort in decreasing order
ii <- order(x) # create the indices which sort x
x[ii] # => sorted
log(x) # component-wise logarithms
x^2 # component-wise squares
sum(x) # sum all numbers
cumsum(x) # compute the *cumulative* sum
prod(x) # multiply all numbers
seq(1, 7, by = 2) # 1, 3, 5, 7
rep(1:3, each = 3, times = 2) # 1 1 1 2 2 2 3 3 3  1 1 1 2 2 2 3 3 3
tail(x, n = 1) # get the last element of a vector
head(x, n = -1) # get all but the last element

## Logical vectors
logical(0) # the empty logical vector
(ii <- x >= 3) # logical vector indicating whether each element of x is >= 3
x[ii] # use that vector to index x => pick out all values of x >= 3
```

```
!ii # negate the logical vector
all(ii) # check whether all indices are TRUE (whether all x >= 3)
any(ii) # check whether any indices are TRUE (whether any x >= 3)
ii |  !ii # vectorized logical OR
ii &  !ii # vectorized logical AND
ii || !ii # logical OR applied to all values
ii && !ii # logical AND applied to all values
3 * c(TRUE, FALSE) # TRUE is coerced to 1, FALSE to 0
class(NA) # NA = 'not available' is 'logical' as well (used for missing data)
z <- 1:3; z[5] <- 4 # two statements in one line (';'-separated)
z # => 4th element 'not available' (NA)
(z <- c(z, NaN, Inf)) # append NaN and Inf
class(z) # still numeric
is.na(z) # check for NA or NaN
is.nan(z) # check for just NaN
is.infinite(z) # check for +/-Inf
z[(!is.na(z)) &  is.finite(z) &  z >= 2] # indexing: pick out all numbers >= 2
z[(!is.na(z)) && is.finite(z) && z >= 2] # watch out (indexing by 'FALSE' => empty vector)

## Character vectors
character(0) # the empty character vector
x <- "apple"
y <- "orange"
(z <- paste(x, y)) # paste together; use sep = "" or paste0() to paste without space
paste(1:3, c(x, y), sep = " - ") # recycling ("apple" appears again)

## Named vectors
(x <- c("a" = 3, "b" = 2)) # named vector of class 'numeric'
x["b"] # indexing elements by name (useful!)
x[["b"]] # drop the name

## Other types of objects are: arrays (incl. matrices), lists, data frames,
## factors, functions


### Arrays and matrices #######################################################

## Matrices
(A  <- matrix(1:12, ncol = 4)) # watch out, R operates on/fills by *columns*
(A. <- matrix(1:12, ncol = 4, byrow = TRUE)) # fills matrix row-wise
(B <- rbind(1:4, 5:8, 9:12)) # row bind
(C <- cbind(1:3, 4:6, 7:9, 10:12)) # column bind
stopifnot(identical(A, C), identical(A., B)) # check whether the constructions are identical
cbind(1:3, 5) # recycling
(A <- outer(1:4, 1:5, FUN = pmin)) # build a (4, 5)-matrix with (i,j)th element being min{i, j}
## => Lower triangular matrix contains column number, upper triangular matrix contains row number

## Some functions
nrow(A) # number of rows
ncol(A) # number of columns
dim(A) # dimension
diag(A) # 1 2 3 4; diagonal of A
diag(3) # identity (3, 3)-matrix
(D <- diag(1:3)) # diagonal matrix with elements 1, 2, 3
D %*% B # matrix multiplication
B * B # Hadamard product, i.e., element-wise product

## Build a correlation matrix and invert it
L <- matrix(c(2, 0, 0,
              6, 1, 0,
             -8, 5, 3), ncol = 3, byrow = TRUE) # Cholesky factor of the ...
Sigma <- L %*% t(L) # ... real, symmetric, positive definite (covariance) matrix Sigma
standardize <- Vectorize(function(r, c) Sigma[r,c]/(sqrt(Sigma[r,r])*sqrt(Sigma[c,c])))
(P <- outer(1:3, 1:3, standardize)) # construct the corresponding correlation matrix
## Alternatively, this could have been done with Matrix::nearPD(Sigma, corr = TRUE)
## which works slightly differently though (by finding a correlation matrix
## close to the given matrix in the Frobenius norm) and thus gives a different answer.
P.inv <- solve(P) # compute P^{-1}; solve(A, b) solves Ax = b (if b is omitted, it defaults to I, thus leading
to A^{-1})
P %*% P.inv # (numerically close to) I
P.inv %*% P # (numerically close to) I

## Other useful functions
rowSums(A) # row sums
apply(A, 1, sum) # the same
colSums(A) # column sums
apply(A, 2, sum) # the same
```

```
### Control statements (just very quickly) #####################################

## R has if() else, ifelse() (a vectorized version of 'if'), for loops (avoid or
## only use if they don't take much run time), repeat and while (with 'break' to
## exit and 'next' to advance to the next loop iteration)

## ... without going into details, note that even 'if()' is a function, so
## instead of:
x <- 4
if(x < 5) y <- 1 else y <- 0 # y is the indicator whether x < 5
## ... write (the much more readable)
y <- if(x < 5) 1 else 0
## ... or even better
(y <- x < 5) # ... as a logical
y + 2 # ... which is internally again converted to {0,1} in calculations

## Also, loops of the type...
x <- integer(5)
for(i in 1:5) x[i] <- i * i
## ... can typically be avoided by something like
x. <- sapply(1:5, function(i) i * i) # of course we know that this is simply (1:5)^2 which is even faster
stopifnot(identical(x, x.))


### Using implemented distributions ############################################

## Probability distributions (d/p/q/r*)
dexp(1.4, rate = 2) # density f(x) = 2*exp(-2*x)
pexp(1.4, rate = 2) # distribution function F(x) = 1-exp(-2*x)
qexp(0.3, rate = 2) # quantile function F^-(y) = -log(1-y)/2
rexp(4,   rate = 2) # draw random variates from Exp(2)



####################################################################################################
##Solutions of the exercises in R course Bologna - Chapter Introduction to R
####################################################################################################

############################
#1.3 Exercises
############################
#1(a).
set.seed(1)
X <- sample(1:6,100,replace = TRUE)
Y <- sample(1:6,10000,replace = TRUE)

#1(b). barplot
TX=table(X)
TY=table(Y)
par(mfrow=c(2,1))
barplot(TX)
barplot(TY)
#Important to first do table: count for every possible outcome how many times it occurred.
#Largest sample closer to population distribution

#1(c).
library(PerformanceAnalytics) #skewness and kurtosis

Z=X
mean(Z)
median(Z)
var(Z)
sd(Z)
IQR(Z)
mad(Z,constant=1)
#by default mad is multiplied with 1.4 (consistency factor for AN distribution)
#normal consistent estimate of standard deviation using IQR: IQR/1.349
skewness(Z)
kurtosis(Z)

#rerun with Z=Y
#population:  3.5; 3.5; 2.9167; 1.7078; 3; 1.5; 0; -1.2686

#2.

1-punif(12,0,20); punif(12,0,20,lower.tail=F)
#by default lower.tail=TRUE and hence calculating probability that is lower than or equal to X (=CDF)
```

```
punif(12,0,20)-punif(4,0,20)
dunif(7,0,20)
dunif(21,0,20)
qunif(0.99,0,20)
qunif(0.5,0,20)
#Median=0.5 quantile
max(runif(n=10,0,20))
max(runif(n=1000,0,20))


#3.
qnbinom(0.9,size=5,prob=0.7)
1-ppois(4,lambda=2); ppois(4,lambda=2,lower.tail=F)
#discrete distribution, hence P(X>=5)=P(X>4)=1-P(X<=4)
pnorm(5,mean=10,sd=4)
dbeta(0.7,5,2)
1-pnorm(log(5),1,2); 1-plnorm(5,1,2)
#P(exp(X)>5)=P(X>log(5))
#or work with lognormal distribution (parameters that you give are mean and standard deviation of normal
distribution, hence of the log of the distribution!)
pgamma(8,0.5,4)
qf(0.95,5,8)
qt(0.99,500)
qnorm(0.99)

#4.
par(mfrow=c(1,1))
x=seq(-3,3,0.01)
#consider lots of points on interval [-3,3]
y=dnorm(x)
plot(x,y,type="l",xlab="x",ylab=expression(phi(x)),main="pdf standard normal")

#5.
x=seq(-4,4,0.01)
y1=pnorm(x,0,1)
y2=pnorm(x,0,sqrt(2))
#second parameter that you give is standard deviation (not the variance)
plot(x,y1,type="l",xlab="x",ylab="F(x)", main="cdf of normal distributions")
lines(x,y2,col="red",lty=3)
#you add cdf to figure that is already open
legend("topleft",legend=c("cdf of N(0,1)","cdf of N(0,2)"),col=c("black","red"),lty=c(1,3))

#6.
set.seed(1)
lambda=2
E1=rexp(5000,lambda)
E2=-log(1-runif(5000))/lambda
#PIT: see slides (integral transformation)
par(mfrow=c(2,2))
plot(density(E1),xlim=c(0,5))
plot(density(E2),xlim=c(0,5))
hist(E1,xlim=c(0,5))
hist(E2,xlim=c(0,5))
par(mfrow=c(1,1))


#7. Central limit theorem

k=2500
n=100

gem=rep(0,k)
set.seed(4)
for(i in 1:k){
  gem[i]=mean(rnorm(n,mean=10,sd=3))
}
par(mfrow=c(2,2))
hist(gem)
plot(density(gem))
gemF=ecdf(gem)
plot(gemF,verticals=T,do.points=F)
qqnorm(gem)
qqline(gem)
par(mfrow=c(1,1))
mean(gem)
sd(gem)
```

```
###################################################################################################
## Data set provided in Gisler (slides Aggregate Loss Modeling)
###################################################################################################



library(MASS)
T=10
vt=c(240755,255571,269739,281708,306888,320265,323481,334753,340265,344757)
Nt=c(13153,14186,14207,13461,21261,19934,15796,15157,17483,19185)
x=Nt/vt
data=round(x*100,2)

t=1982:1991
plot(t,x, xlab="time", ylab="observed frequencies")
abline(h=mean(x))

#Poisson
lambda=fitdistr(x,"Poisson")
lambda

vco=(lambda$estimate*mean(vt))^(-1/2)
vco

abline(h=lambda$estimate+lambda$estimate*vco,col="green")
abline(h=lambda$estimate-lambda$estimate*vco,col="green")


#goodness-of-fit
X2=sum(vt*(Nt/vt-lambda$estimate)^2)/lambda$estimate
X2

qchisq(0.99,T-1)

#X2>>qchisq hence reject H0


# library(vcd)
# gf=goodfit(Nt,type="poisson", method="MinChisq")
# summary(gf)
# plot(gf)

#Negative-binomial
#fitdistr(x,"negative binomial")

lambda=1/(sum(vt))*sum(Nt)
lambda
Vt=1/(T-1)*sum(vt*(Nt/vt-lambda)^2)
Vt
#larger than 5.43
gamma=lambda^2/(Vt-lambda)*1/(T-1)*(sum(vt)-sum(vt^2)/sum(vt))
gamma

vco=sqrt(1/(lambda*mean(vt))+1/gamma)
vco

#lambda estimate same as before!
plot(t,x, xlab="time", ylab="observed frequencies")
abline(h=mean(x))
abline(h=lambda+lambda*vco,col="green")
abline(h=lambda-lambda*vco,col="green")




###################################################################################################
## Exercise Natural Hazards in Switzerland (slides Aggregate Loss Modeling)
###################################################################################################

Y=c(52.8,135.2,55.9,138.6,122.9,55.8,368.2,83.8,78.5,75.3,178.3,182.8,54.4,365.3,1051.1)
sum(log(Y))
#a)
alpha_M=1/(mean(log(Y))-log(50))*14/15
alpha_M
alpha_bias=15/14*alpha_M
alpha_bias
```

```
#b)
alfa=alpha_M
#alfa=alpha_bias
15/20*((1-(2000/50)^(-alfa+1))*(50*alfa)/(alfa-1)+2000*(2000/50)^(-alfa))

15/20*1/(alfa-1)*(alfa*50-50^alfa*2000/(2000^alfa))

15/20*50/(alfa-1)*(alfa-(50/2000)^(alfa-1))


#c)
(2000/50)^(-alfa)


(2*10^9/10^6/50)^(-alpha_M)
```