```r
# 3.1 Programming your own model

# In this session, our interest is in being able to estimate our own models
# without the need for packages.

#==============================================================================>
# OLS with heteroskedasticity-robust standard errors
# For ease of exposition, let us load the data dat.csv and run our OLS as in
# 1.1. Recall to set the working directory to the workshop folder.

## Loading data into R
# Let us load the .csv data from the working directory. This is the mroz data
# is taken from the Jeffery Wooldridge's textbook website

dat<- read.csv("dat.csv",header = T,sep = " ")
names(dat); dim(dat) # check column names and dimension of matrix
nr = dim(dat)[[1]] # extract number of rows
nc = dim(dat)[[2]] # extract number of columns
#---------------------------------------------->
## Manipulating the data set
# create a matrix of two columns age and experience
xx = as.matrix(cbind(dat$education,dat$experience,dat$age,dat$age^2))
#matrix of explanatory variables; # add quadratic term of age

# Obtain the regression object:
reg<- lm(dat$nonwife~xx)
u <- reg$residuals #obtain the residuals from our object
summary(u) #summary of residuals

# create matrix of explanatory variables (including intercept term)
x = as.matrix(cbind(1,xx))

# Compute heteroskedasticity-robuts standard errors
u2 <- u^2 # vector of squared residuals
XDX <- 0 #initialise the sandwich term

## Here one needs to calculate X'DX. But due to the fact that
## D is huge (NxN), it is better to do it with a cycle. We use a loop


for(i in 1:nrow(x)) {
  XDX <- XDX + u2[i]*x[i,]%*%t(x[i,])
}

# inverse(X'X)
XX1 <- solve(t(x)%*%x)

# Variance calculation (Bread x meat x Bread)
varcovar <- XX1 %*% XDX %*% XX1

# degrees of freedom adjustment
dfc <- sqrt(nrow(x))/sqrt(nrow(x)-ncol(x))

# Standard errors of the coefficient estimates are the
# square roots of the diagonal elements
(stdh <- dfc*sqrt(diag(varcovar)))
```

```r
# compare with OLS standard errors
sqrt(diag(vcov(reg)))
# How do the above compare?

#==============================================================================>
# Minimising the sum of squares (one parameter case)
set.seed(21);xx = runif(200,-1,3) #generate values

fn1 = function(t) sum((xx-t)^2); fn1=Vectorize(fn1)

tt = seq(-2,4,length.out = 2000) #create a grid of t values

plot(tt,fn1(tt),type = "l") #plot t values agains the sum of squared errors
# we suspect a minimum is around 1. This helps us in the choice of starting values

opt = optimise(fn1,c(0,2)) #This is a one-parameter minimisation function in R.
opt


print(noquote(paste("The minimum of",opt$objective,"is attained at",opt$minimum)))

#==============================================================================>
#Maximum Likelihood estimation by R

# Let us create a data set
# Input the data.

X <- c(rep(0,14), rep(1,30), rep(2,36), rep(3,68), rep(4, 43), rep(5,43),
       rep(6, 30), rep(7,14), rep(8,10), rep(9, 6), rep(10,4), rep(11,1),
       rep(12,1))

length(X)
# If we suppose Poisson model might be a good model for this dataset,
# we still need to find out which Poisson, that is estimate the parameter λ
# in the Poisson model:

#Plot out the histogram of the data

hist(X, main="histogram of number of right turns", right=FALSE, prob=TRUE)
n <- length(X) #obtain length of the vector X

# Supposing the poisson density function, we can define the negative log
# likelihood function as follows:

negloglike<-function(lam) {
  n*lam - sum(X)*log(lam) + sum(log(factorial(X)))
  }

#Once the function is defined in R, you can evaluate the function value by
# giving it an input value for lam. For example, you can type in

negloglike(0.3) #returns a function value

# Since this function is in one dimension, let us plot it.
negloglike<- Vectorize(negloglike) # vectorise the function, why?
curve(negloglike,from = 2, to = 6.5)
#After we define the negative log likelihood, we can perform the optimization as
```

```r
# following:

(out<-optimise(negloglike,c(2,6.5)))

#=============================================================================>
#Exercise: Fitting a gamma distribution to the nonwife variable.

mydata = dat$nonwife[dat$nonwife>0] #choose positive values of this variable
# only one (probably pathological) observation is dropped

fix(mydata) #visualise data

#plot the density of a distribution, use the following:
plot(density(mydata)) # a density plot


logbeta = function(theta,mydata){
  a=theta[1]
  b=theta[2]
  n=length(mydata)
  -n*(a*log(b)-log(gamma(a))) - (a-1)*sum(log(mydata))+b*sum(mydata)
} #return negative of the log-likelihood

start=c(1,1) # proposed starting values
logbeta(theta=start,mydata=mydata)
#log likelihood function value evaluated at starting values

output = optim(start, fn=logbeta, mydata=mydata, hessian = TRUE)
# optimise using using optim
output

#=============================================================================>
# Fitting mydata to a caucy distribution.

cau.fn<- function(parm,dat){
  mu = parm[1]; sig = parm[2]
  val = 0
  for(i in 1:length(dat)){ #check to make sure data is a vector
    val = val + (-log(pi)+log(sig)-log(sig^2+(dat[i]-mu)^2))
  }
  -val # negative of log-likelihood for minimisation
}
# use mean and variance as starting values.
(start = c(18,10)) #proposed starting values
cau.fn(parm = start,dat = mydata) #compute function value at start


(opt<- optim(par = start, fn=cau.fn,dat = mydata))

plot(mydata,dcauchy(mydata,location = opt$par[1],scale = opt$par[2]))
# plot of fitted cauchy density

lines(density(mydata)) #add plot of empirical density
# how do they compare?

#=============================================================================>
```

```r
## Coding quantile reg likelihood with asymmetric laplace distribution

# a convenient form of the check function
rfn<- function(u,tau) (abs(u) + (2*tau - 1)*u)*0.5
#example:
rfn(0.5,0.25); rfn(-0.5,0.25) # it is asymmetric

# asymmetric laplace likelihood function
alfn<- function(pars,tau,y,x){
  rrfn<- function(u) rfn(u,tau=tau)
  rrfn=Vectorize(rrfn)
  N = length(y); x = cbind(1,x)
  pars = matrix(pars,ncol = 1)
  uu = y - x%*%pars # residuals
  val = -sum(rrfn(uu)) # the log-likelihood
  return(-val) #return negative log-likelihood for minimisation
}

# with our data set dat:
X = x[,-1]
y = dat$nonwife
tau = 0.5
pars=unname(coef(reg))
alfn(pars = pars,tau = tau,y=y,x=X) #test function value

(ans0=optim(par = pars,fn=alfn,tau=0.5,y=y,x=X))
#obtain parameters using optim()

# changing the coefficient of skewness:
(ans1=optim(par = pars,fn=alfn,tau=0.25,y=y,x=X))
(ans2=optim(par = pars,fn=alfn,tau=0.75,y=y,x=X))

# using the quantile regression function rq()
require("quantreg")
rq0=rq(y~X,0.5)
rq1=rq(y~X,0.25)
rq2=rq(y~X,0.75)

# comparing to values delivered in the rq function
rbind(ans0$par,rq0$coefficients)
rbind(ans1$par,rq1$coefficients)
rbind(ans2$par,rq2$coefficients)
#==============================================================================>

# Let us write a general function for the L1 penalisation of objective function

l1pen<- function(pars,objfn,lam,...){
  objfn(pars,...) + lam*(sum(abs(pars)))
}
# A general function that takes as input, parameter values, objective function,
# the penalisation term lambda, and any other extra input,...

# Discuss the advantage of ... in a function call

# Example:
#----------------------------------------------->
# Penalising the cauchy model
```

```r
(start = c(18,10)) #proposed starting values
l1pen(pars = start,objfn = cau.fn,dat = mydata,lam = 4)

# optimisation of the l1-penalised term:

l1obj1<- optim(par = start,fn=l1pen, objfn=cau.fn, dat = mydata,
               lam = 4,hessian = T)
l1obj1

# Increase the penalty and see what happens:
l1obj2<- optim(par = start,fn=l1pen, objfn=cau.fn, dat = mydata,
               lam = 24,hessian = T)
l1obj2

l1obj3<- optim(par = start,fn=l1pen, objfn=cau.fn, dat = mydata,
               lam = 84,hessian = T)
l1obj3

l1obj4<- optim(par = start,fn=l1pen, objfn=cau.fn, dat = mydata,
               lam = 120,hessian = T)
l1obj4

# what do you observe as lam increases?
rbind(l1obj1$par,l1obj2$par,l1obj3$par,l1obj4$par)


#----------------------------------------------->
# Penalising the asymmetric laplace (quantile) model
l1lap0<- optim(par = pars,fn=l1pen, objfn=alfn, tau=0.25,y=y,x=X,
               lam = 0)
l1lap0 # this is the unconstrained case, why?


l1lap1<- optim(par = pars,fn=l1pen, objfn=alfn, tau=0.25,y=y,x=X,
               lam = 5)
l1lap1

l1lap2<- optim(par = pars,fn=l1pen, objfn=alfn, tau=0.25,y=y,x=X,
               lam = 15)
l1lap2

l1lap3<- optim(par = pars,fn=l1pen, objfn=alfn, tau=0.25,y=y,x=X,
               lam = 35)
l1lap3

l1lap4<- optim(par = pars,fn=l1pen, objfn=alfn, tau=0.25,y=y,x=X,
               lam = 75)
l1lap4
#----------------------------------------------->

# Exercises:
# Let us try some exercises from the attached PDF text
```