Computational Statistics. Chapter 1: Continuous optimization. Solution of exercises

Thierry Denoeux

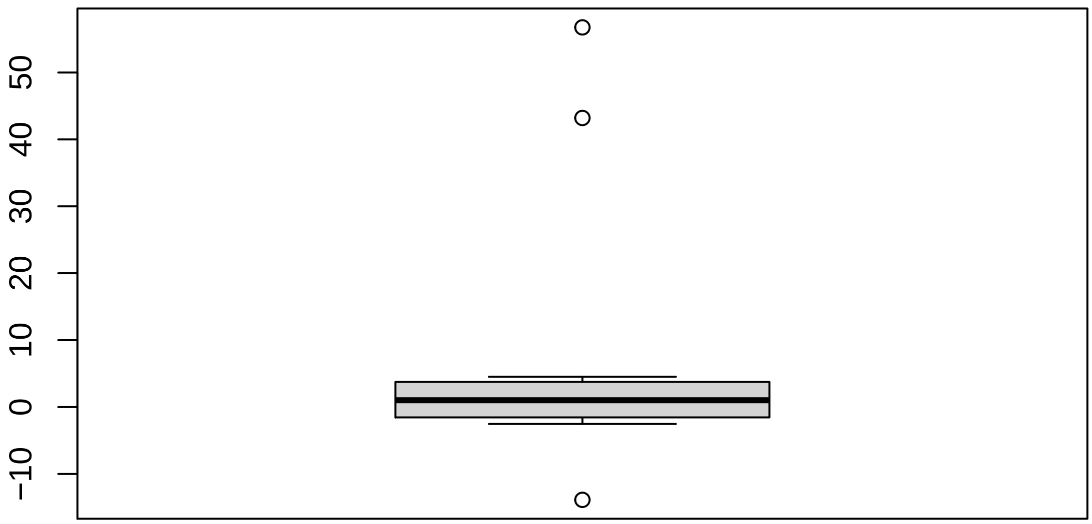
8/18/2021

Exercise 1

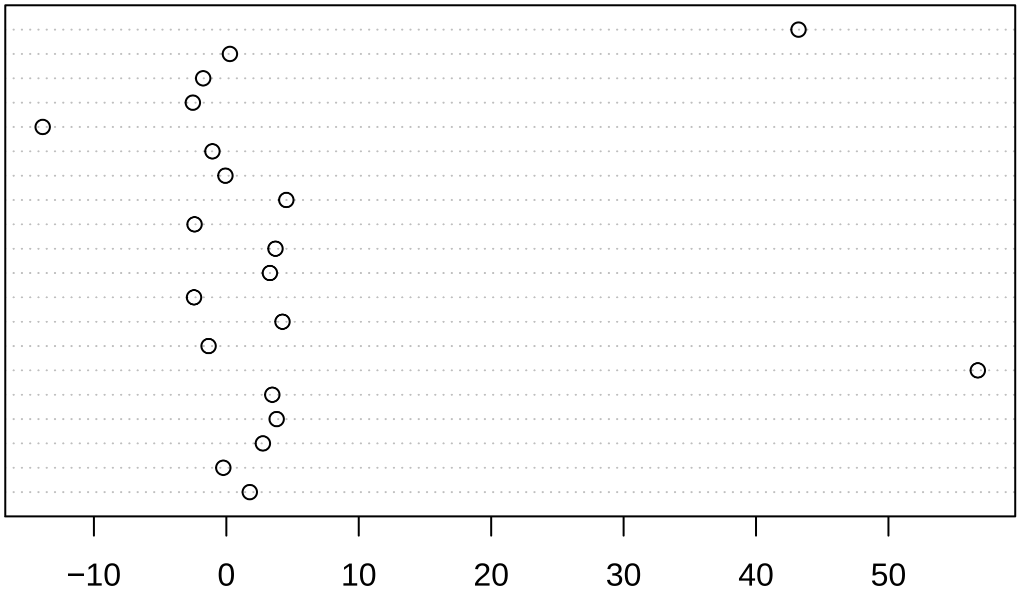
Question a

```
x<-c(1.77,-0.23,2.76,3.80,3.47,56.75,-1.34,4.24,-2.44,
     3.29,3.71,-2.40,4.53,-0.07,-1.05,-13.87,-2.53,
     -1.75,0.27,43.21)
n<- length(x)

boxplot(x)
```




```
dotchart(x)
```



Question b

We first write a function to compute the log-likelihood:

```
loglik <- function(theta,x) return(sum(log(dcauchy(x,location=theta))))
```

We compute the log-likelihood for different values of θ:

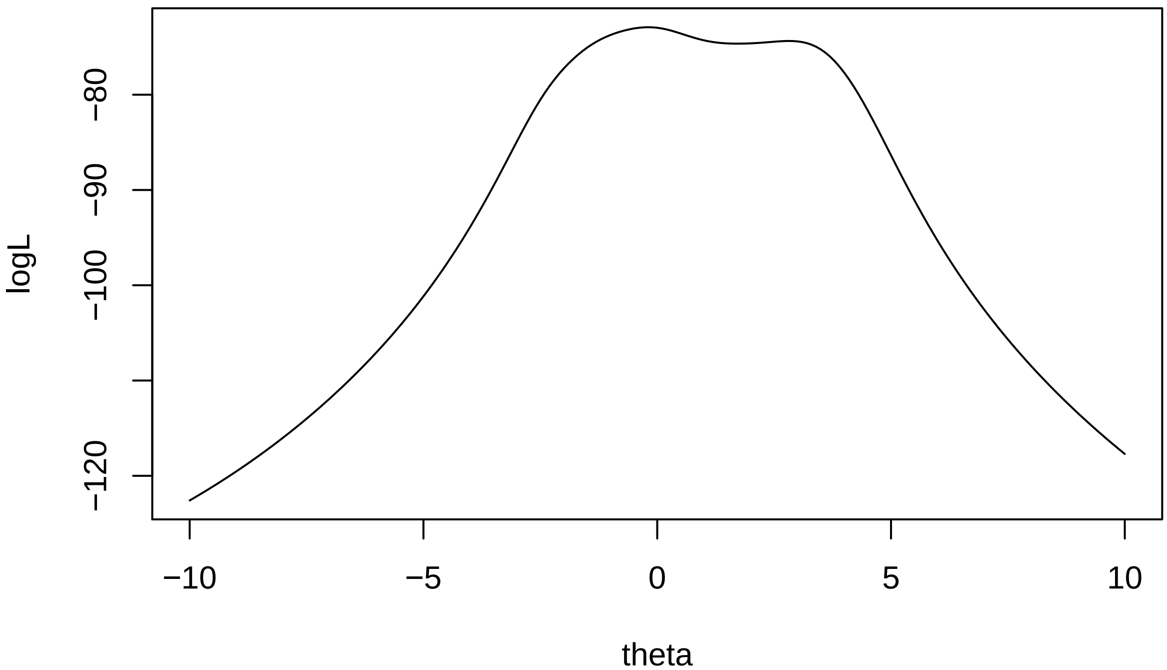
```
theta<- seq(-10,10,0.1)
N<-length(theta)
logL<-rep(0,N)
for(i in 1:N) logL[i]<- loglik(theta[i],x)
```

We can get the same result much faster without a loop, thanks to function `sapply`:

```
logL<-sapply(theta,loglik,x)
```

Finally, we plot the result:

```
plot(theta,logL,type="l")
```



We observe that the likelihood has 2 modes.

Question c

We first need to compute the score function (first derivative of the log-likelihood). We have

$$L(\theta) = \frac{1}{\pi^n}\prod_{i=1}^{n}\frac{1}{(x_i-\theta)^2+1}$$

$$\ell(\theta) = -\sum_{i=1}^{n}\log[(x_i-\theta)^2+1] - n\log\pi$$

$$\ell'(\theta) = 2\sum_{i=1}^{n}\frac{x_i-\theta}{(x_i-\theta)^2+1}$$

We can then write the R function:

```
dloglik <- function(theta,x) return(2*sum((x-theta)/((x-theta)^2+1)))
```

This is a function that encodes the bisection method:

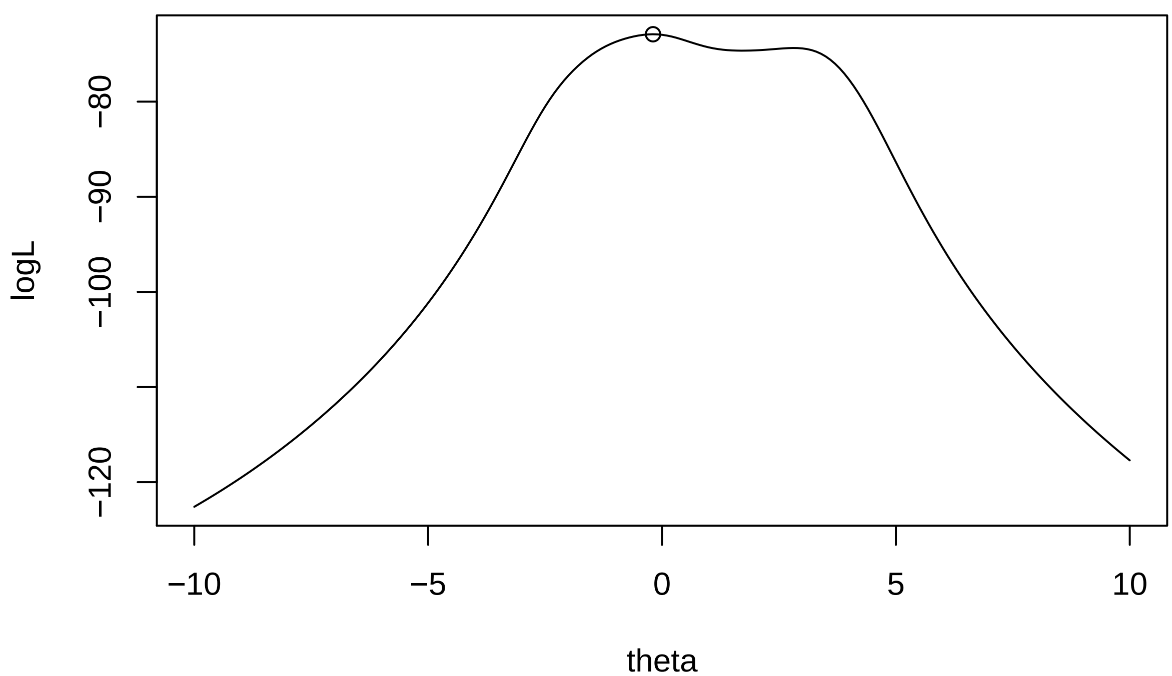
```
bisection <-function(fun,dfun,a,b,epsi,...){
    theta<-(a+b)/2
    delta<-1
    while(delta>epsi){
        theta0<-theta
        if(dfun(a,x)*dfun(theta0,x)<=0) b<-theta0 else a<-theta0
        theta<-(a+b)/2
        delta<-abs(theta-theta0)/abs(theta0)
        print(c(a,b,delta))
    }
    return(list(objective=fun(theta,x),optimum=theta))
}
```

We run it on the data and plot the result:

```
opt<-bisection(loglik,dloglik,-1,3,1e-6,x)
```

```
## [1] -1  1  1
## [1]  -1   0 Inf
## [1] -0.5  0.0  0.5
## [1] -0.25  0.00  0.50
## [1] -0.250 -0.125  0.500
## [1] -0.2500000 -0.1875000  0.1666667
## [1] -0.21875000 -0.18750000  0.07142857
## [1] -0.20312500 -0.18750000  0.03846154
## [1] -0.1953125 -0.1875000  0.0200000
## [1] -0.19531250 -0.19140625  0.01020408
## [1] -0.193359375 -0.191406250  0.005050505
## [1] -0.192382812 -0.191406250  0.002538071
## [1] -0.192382812 -0.191894531  0.001272265
## [1] -0.192382812 -0.192138672  0.000635324
## [1] -0.1923828125 -0.1922607422  0.0003174603
## [1] -0.1923217773 -0.1922607422  0.0001586798
## [1] -1.922913e-01 -1.922607e-01  7.935248e-05
## [1] -1.922913e-01 -1.922760e-01  3.967939e-05
## [1] -1.922913e-01 -1.922836e-01  1.983891e-05
## [1] -1.922874e-01 -1.922836e-01  9.919257e-06
## [1] -1.922874e-01 -1.922855e-01  4.959678e-06
## [1] -1.922874e-01 -1.922865e-01  2.479827e-06
## [1] -1.922870e-01 -1.922865e-01  1.239910e-06
## [1] -1.922867e-01 -1.922865e-01  6.199559e-07
```

```
plot(theta,logL,type="l")
points(opt$optimum,opt$objective)
```



Question d

Let us program Newton's method. For that, we need the second derivative of the log-likelihood:

$$\ell''(\theta) = 2\sum_{i=1}^{n} \frac{(x_i-\theta)^2-1}{[(x_i-\theta)^2+1]^2}$$

We write the corresponding R function:

```
d2loglik <- function(theta,x) return(2*sum(((x-theta)^2-1)/((x-theta)^2+1)^2))
```

This is an implementation of Newton's method:

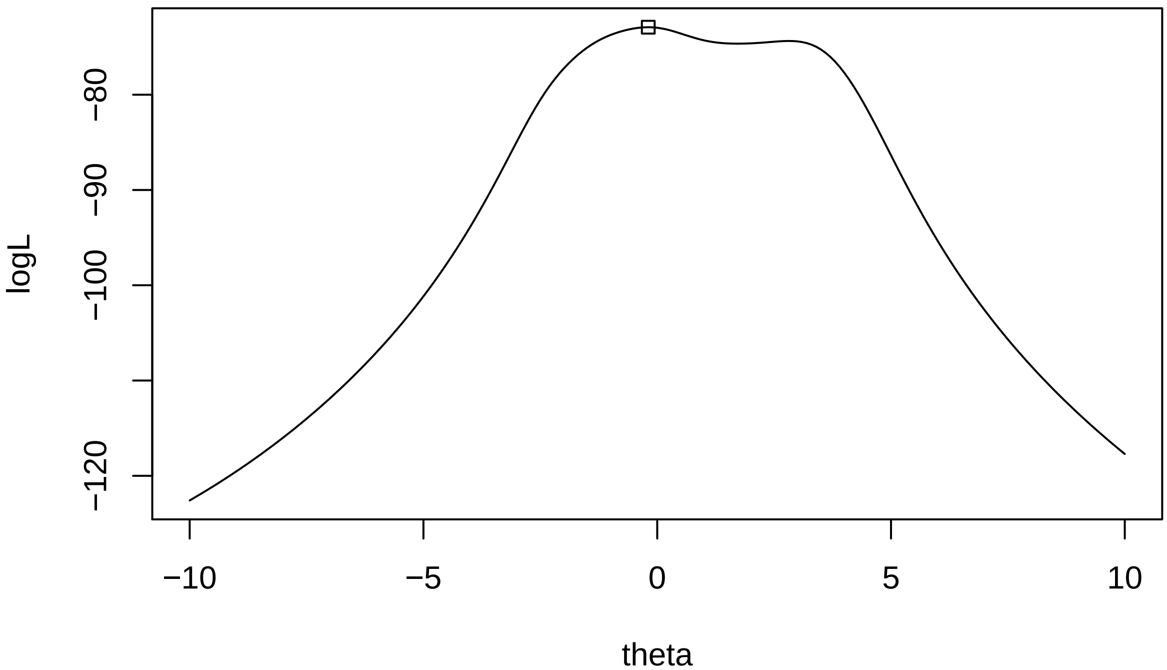
```
newton <-function(fun,dfun,d2fun,theta0,epsi,tmax,...){
    delta=1
    t<-0
    while((delta>epsi)&(t<=tmax)){
      t<-t+1
        theta<-theta0-dfun(theta0,x)/d2fun(theta0,x)
        delta<-abs(theta-theta0)/abs(theta0)
        obj<-fun(theta,x)
        print(c(t,theta,obj,delta))
        theta0<-theta
    }
    return(list(objective=obj,optimum=theta,
                derivative=dfun(theta,x),
                derivative2=d2fun(theta,x)))
}
```

We run it on our data and plot the results:

```
theta0<- 0
opt=newton(loglik,dloglik,d2loglik,theta0,1e-6,1000,x)
```

```
## [1]   1.0000000  -0.1963366 -72.9158449         Inf
## [1]   2.00000000  -0.19228252 -72.91581962   0.02064858
## [1]  3.000000e+00 -1.922866e-01 -7.291582e+01  2.126316e-05
## [1]  4.000000e+00 -1.922866e-01 -7.291582e+01  2.109094e-11
```

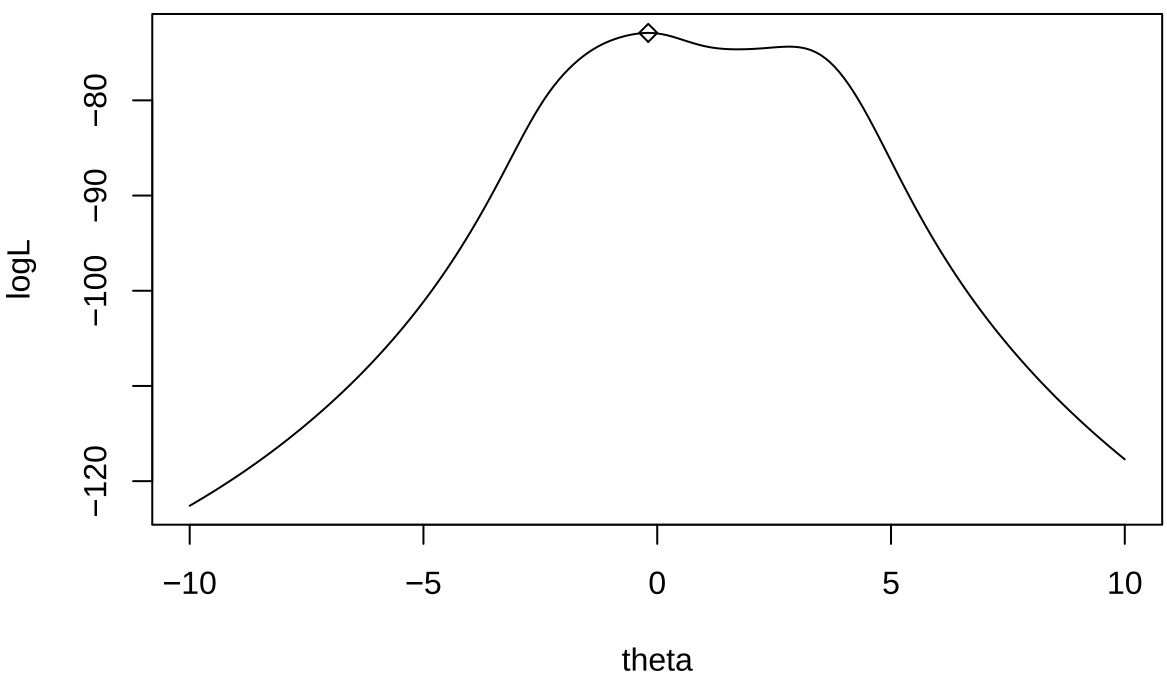
```
plot(theta,logL,type="l")
points(opt$optimum,opt$objective,pch=22)
```



Question e

Finally, we can get the same result with the R built-in function `optimize`:

```
opt <-optimize(f=loglik,lower=-2,upper=2,maximum=TRUE,x=x)
plot(theta,logL,type="l")
points(opt$maximum,opt$objective,pch=23)
```



Exercise 2

Newton's method

For Newton's method, we need to compute the log-likelihood as well as its first and second derivatives. We have

$$L(\theta) = \prod_{i=1}^{n} \frac{\theta^{x_i}}{x_i[-\log(1-\theta)]}$$

$$\ell(\theta) = \log\theta \sum_{i=1}^{n} x_i - \sum_{i=1}^{n} \log(x_i) - n\log[-\log(1-\theta)]$$

$$\ell'(\theta) = \frac{\sum_{i=1}^{n} x_i}{\theta} + \frac{n}{(1-\theta)\log(1-\theta)}$$

$$\ell''(\theta) = -\frac{\sum_{i=1}^{n} x_i}{\theta^2} + n\frac{\log(1-\theta)+1}{(1-\theta)^2[\log(1-\theta)]^2}$$

These functions can be encoded as follows:

```
loglik1<-function(theta,x){ # loglikelihood
  n<-length(x)
  return(sum(x)*log(theta)-n*log(-log(1-theta)) -sum(log(x)))
}
dloglik1<-function(theta,x){  # first derivative
  n<-length(x)
  return(sum(x)/theta + n/((1-theta)* log(1-theta)))
}
d2loglik1<-function(theta,x){ # second derivative
  n<-length(x)
  return(-sum(x)/theta^2 + n*(1+log(1-theta))/((1-theta)^2* log(1-theta)^2))
}
```

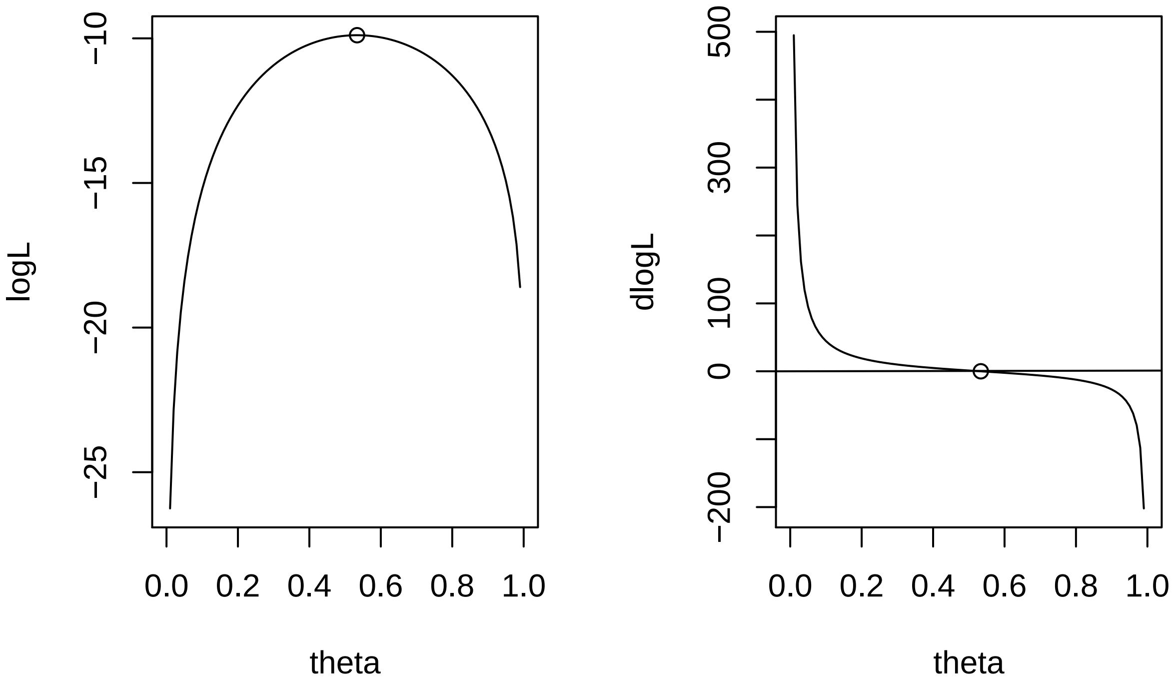
We use the function `newton`from Exercise 1:

```
theta0<- 0.8
x<-c(1, 1, 1, 1, 1, 1, 2, 2, 2, 3)
opt=newton(loglik1,dloglik1,d2loglik1,theta0,1e-6,1000,x)
```

```
## [1]   1.0000000   0.6502650 -10.1271031   0.1871688
## [1]  2.0000000  0.5444798 -9.8929451  0.1626801
## [1]  3.00000000  0.53359344 -9.89093316  0.01999413
## [1]  4.000000e+00  5.335892e-01 -9.890933e+00  7.890585e-06
## [1]  5.000000e+00  5.335892e-01 -9.890933e+00  1.474363e-12
```

We plot the result:

```
theta<- seq(0,1,0.01)
logL<-sapply(theta,loglik1,x)
dlogL<-sapply(theta,dloglik1,x)
par(mfrow=c(1,2))
plot(theta,logL,type="l")
points(opt$optimum,opt$objective)
plot(theta,dlogL,type="l")
points(opt$optimum,0)
abline(0,1)
```




```
par(mfrow=c(1,1))
```

Fisher scoring

To implement the Fisher scoring method, we need to compute the Fisher information. We have

$$I_n(\theta) = -\mathbb{E}_\theta[\ell''(\theta)] = \frac{n\mathbb{E}_\theta[X]}{\theta^2} - n\frac{\log(1-\theta)+1}{(1-\theta)^2[\log(1-\theta)]^2},$$

with

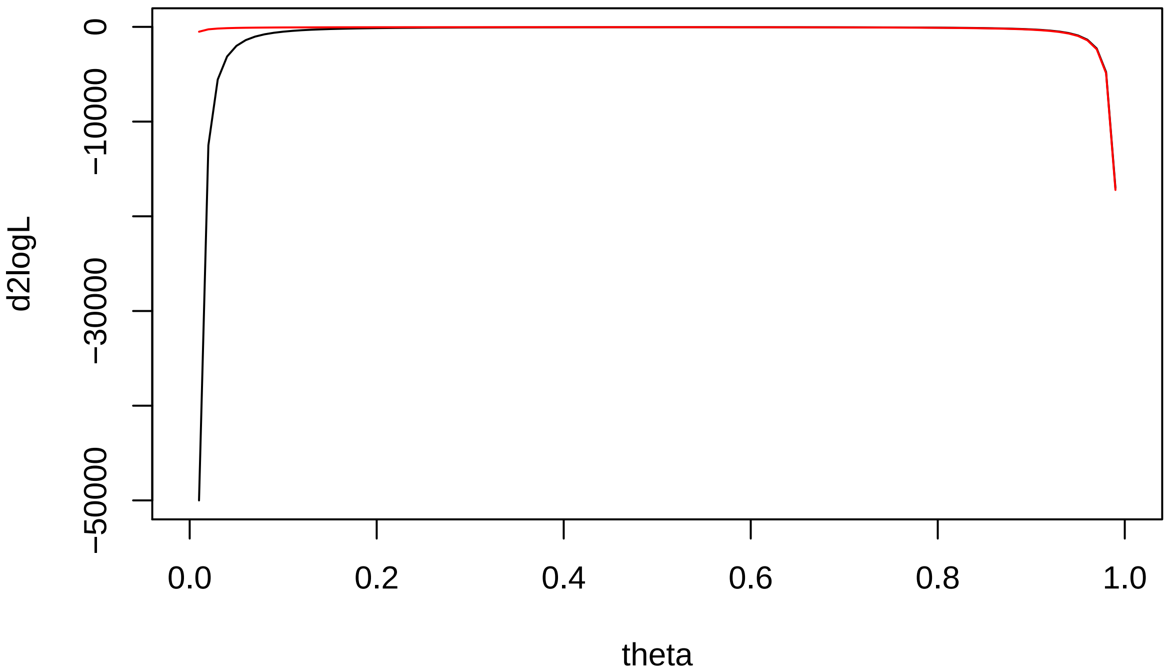
$$\mathbb{E}_\theta[X] = \frac{-\theta}{(1-\theta)\log(1-\theta)}.$$

The following function computes $-I_n(\theta)$:

```
fisher.info <-function(theta,x){ # Fisher information with minus sign
  n<-length(x)
  EX<- -1/log(1-theta) * theta/(1-theta)
  return(-n*EX/theta^2 + n*(1+log(1-theta))/((1-theta)^2* log(1-theta)^2))
}
```

We can check that $I_n(\theta) \approx -\ell''(\theta)$, especially around $\widehat{\theta}$:

```
d2logL<-sapply(theta,d2loglik1,x)
fish<-sapply(theta,fisher.info,x)
plot(theta,d2logL,type='l')
lines(theta,fish,col="red")
```



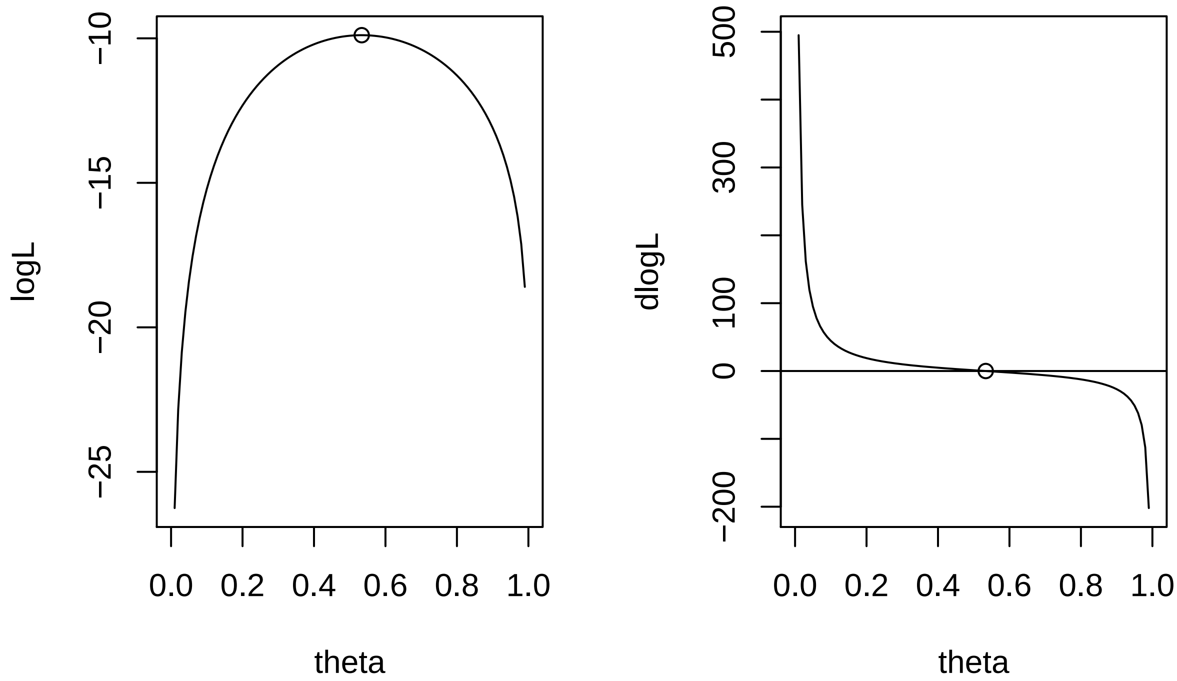
To use the Fisher scoring method, we can use function `newton` and pass minus the Fisher information instead of the second derivative as argument:

```
theta0<- 0.8
opt=newton(loglik1,dloglik1,fisher.info,theta0,1e-6,1000,x)
```

```
## [1]   1.0000000   0.6738722 -10.2362220   0.1576598
## [1]  2.0000000  0.5709805 -9.9146869  0.1526872
## [1]  3.00000000  0.53617188 -9.89104631  0.06096289
## [1]  4.00000000  0.53360145 -9.89093316  0.00479404
## [1]  5.000000e+00  5.335892e-01 -9.890933e+00  2.288549e-05
## [1]  6.000000e+00  5.335892e-01 -9.890933e+00  5.113816e-10
```

Plotting the results:

```
par(mfrow=c(1,2))
plot(theta,logL,type="l")
points(opt$optimum,opt$objective)
plot(theta,dlogL,type="l")
points(opt$optimum,0)
abline(h=0)
```




```
par(mfrow=c(1,1))
```

Exercise 3

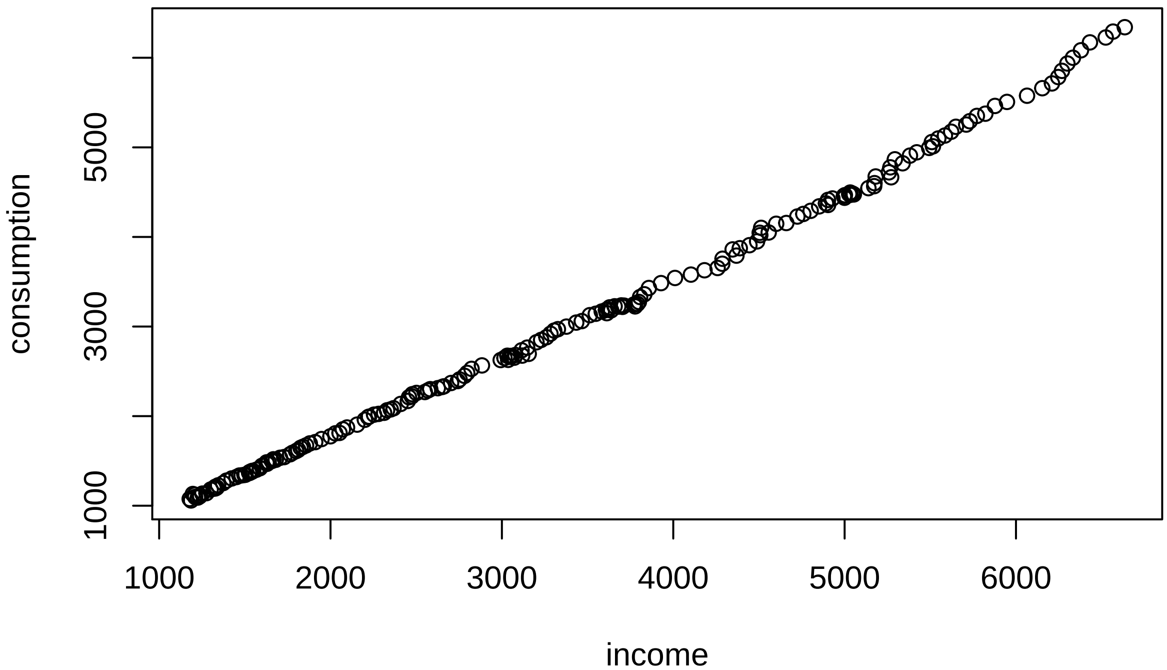
Question a

We start by reading the file:

```
data <- read.table("/Users/Thierry/Documents/R/Data/Compstat/F5_2.txt",header=TRUE)
```

Consumption and income correspond, respectively, to variables `realdpi` and `realcons`. We plot these two variables:

```
plot(data$realdpi,data$realcons,xlab="income",ylab="consumption")
```



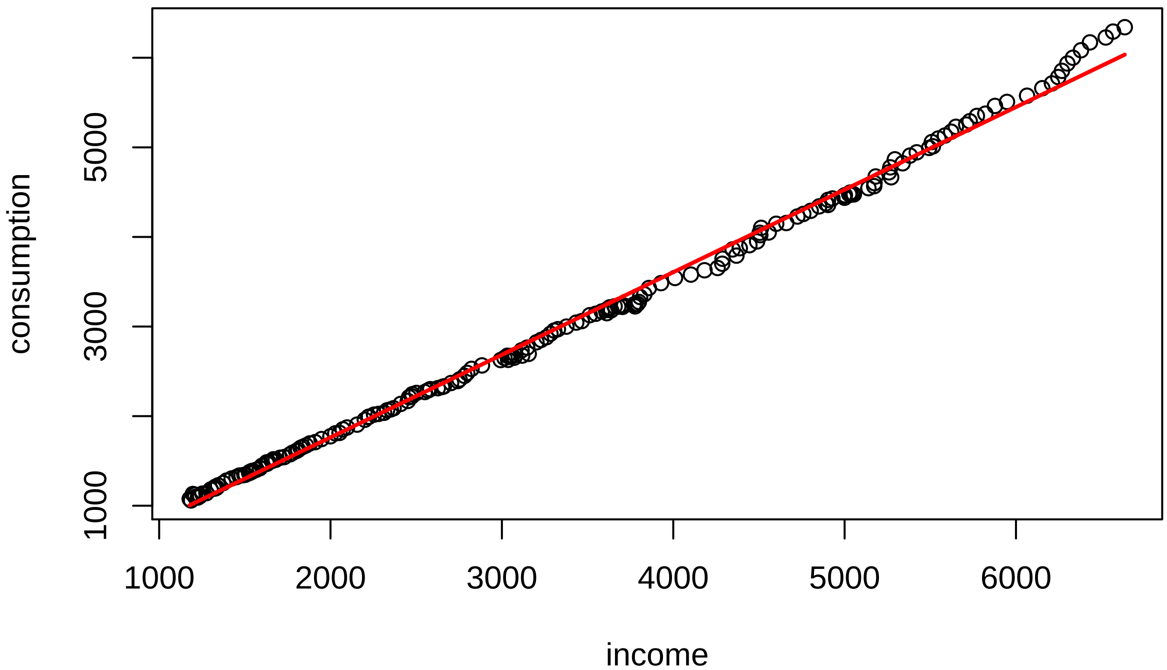
The relationship between consumption and income seems approximately linear. We then estimate α and β using linear regression and display a summary of the result:

```
reg<-lm(realcons ~ realdpi, data=data)
summary(reg)
```

```
##
## Call:
## lm(formula = realcons ~ realdpi, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -191.42  -56.08    1.38   49.53  324.14
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -80.354749  14.305852  -5.617 6.38e-08 ***
## realdpi       0.921686   0.003872 238.054  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 87.21 on 202 degrees of freedom
## Multiple R-squared:  0.9964, Adjusted R-squared:  0.9964
## F-statistic: 5.667e+04 on 1 and 202 DF,  p-value: < 2.2e-16
```

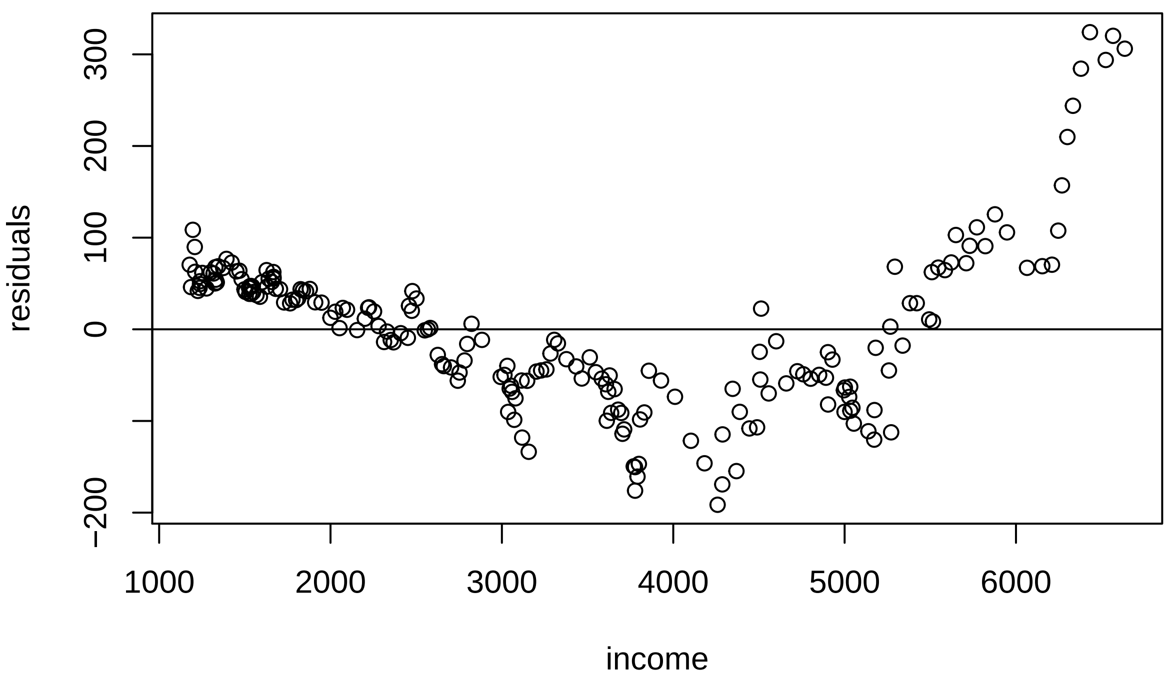
A plot of the least-squares line together with the data seems to indicate a good fit:

```
plot(data$realdpi,data$realcons,xlab="income",ylab="consumption")
lines(data$realdpi,reg$fitted.values,col='red',lwd=2)
```



However, a plot of the residuals vs. income shows that the linear model is misspecified. (The residuals do not appear as purely random, they depend on the income):

```
plot(data$realdpi,reg$residuals,xlab="income",ylab="residuals")
abline(h=0)
```



This analysis justifies the use of nonlinear regression.

Question b

As in the slides, let us denote the response variable (consumption) by Y and the covariate (income) by z. The model can then be written as

$$Y = f(z, \theta) + \epsilon$$

with $f(z,\theta) = \alpha + \beta z^\gamma$ and $\theta = (\alpha, \beta, \gamma)$.

To linearize f around $\theta = \theta^{(t)}$, let us first compute the gradient of f with respect to θ. We have

$$\frac{\partial f}{\partial \alpha} = 1, \quad \frac{\partial f}{\partial \beta} = z^\gamma,$$

and

$$\frac{\partial f}{\partial \gamma} = \frac{\partial(\alpha + \beta \exp(\gamma \log z))}{\partial \gamma} = \beta(\log z)\exp(\gamma \log(z)) = \beta(\log z) z^\gamma.$$

The linear Taylor series expansion of f around $\theta = \theta^{(t)}$ can be written as

$$f(z,\theta) \approx f(z, \theta^{(t)}) + (\theta - \theta^{(t)})^T \mathbf{f}'(\theta^{(t)}).$$

Here we have

$$f(z,\theta) \approx \alpha^{(t)} + \beta^{(t)} z^{\gamma^{(t)}} + (\alpha - \alpha^{(t)}) + (\beta - \beta^{(t)}) z^{\gamma^{(t)}} + (\gamma - \gamma^{(t)})\beta^{(t)}(\log(z)) z^{\gamma^{(t)}}.$$

We can thus write

$$\underbrace{y_i - f(z_i, \theta^{(t)})}_{x_i^{(t)}} \approx (\theta - \theta^{(t)})^T \underbrace{(1, z_i^{\gamma^{(t)}}, \beta^{(t)}(\log z_i) z_i^{\gamma^{(t)}})}_{\mathbf{a}_i^{(t)}}$$

Denoting by $\mathbf{x}^{(t)}$ the vector of length n with components $x_i^{(t)}$ and by $\mathbf{A}^{(t)}$ the $n \times 3$ matrix with ith row $\mathbf{a}_i^{(t)}$, we can write the following regression model:

$$\mathbf{x}^{(t)} = \mathbf{A}^{(t)}(\theta - \theta^{(t)}) + \epsilon.$$

The expression of the ordinary least-squares (OLS) estimate gives us the update equation:

$$\theta^{(t+1)} = \theta^{(t)} + \left((\mathbf{A}^{(t)})^T \mathbf{A}^{(t)}\right)^{-1} (\mathbf{A}^{(t)})^T \mathbf{x}^{(t)}.$$

We will now write a generic implementation of the Gauss-Newton algorithm. The inputs are the vector of responses `y`, the vector or matrix of covariates `z`, the initial parameter value `theta0`, and functions `fun` and `grad` for, respectively, function f and its gradient. Here, we have

```
fun<-function(theta,z) return(theta[1]+theta[2]*z^theta[3])
```

and

```
grad<-function(theta,z) return(cbind(rep(1,length(z)),z^theta[3],theta[2]*log(z)*z^theta[3]))
```

We note that function `grad` returns matrix $\mathbf{A}$. We can now write a generic function for the Gauss-Newton algorithm:

```
gauss.newton<-function(y,z,theta0,fun,grad,epsi=1e-6){
  delta<-1
  LS<-function(theta,y,z) return(sum((y-fun(theta,z))^2)) # Computes the sum of squared errors
  g<-LS(theta0,y,z)
  t<-0
  print(c(t,theta0,g,delta))
  while(delta>epsi){  # Main loop
    t<-t+1
```

```
    x <- y-fun(theta0,z)
    A<-grad(theta0,z)
    theta1<-theta0+solve(t(A)%*%A)%*%t(A)%*%x
    delta<-sqrt(sum((theta1-theta0)^2))/sqrt(sum((theta0)^2))
    g<-LS(theta1,y,z)
    print(c(t,theta1,g,delta))
    theta0<-theta1
  }
  return(list(theta=theta1,g=g))
}
```

Let us now run this algorithm with our data. We initialize the parameter with the OLS estimate:

```
theta0<-c(reg$coefficients,1)
```

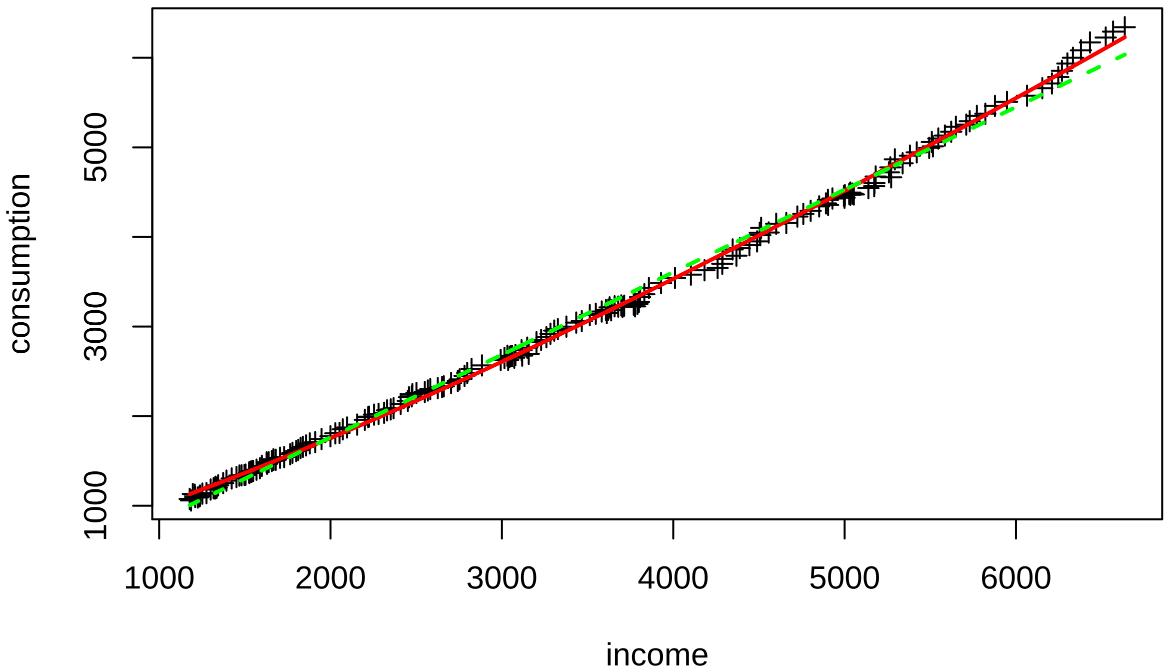
We then run the algorithm:

```
opt<-gauss.newton(y=data$realcons,z=data$realdpi,theta0,fun,grad)
```

```
##                  (Intercept)       realdpi
##   0.000000e+00 -8.035475e+01  9.216857e-01  1.000000e+00  1.536322e+06
##
##   1.000000e+00
## [1]  1.000000e+00  5.646368e+02 -1.037906e+00  1.232968e+00  1.847814e+11
## [6]  8.025689e+00
## [1] 2.000000e+00 4.582590e+02 1.006556e-01 1.231731e+00 2.040661e+07
## [6] 1.884105e-01
## [1] 3.000000e+00 4.582245e+02 1.005646e-01 1.245968e+00 5.817015e+05
## [6] 8.155484e-05
## [1] 4.000000e+00 4.588704e+02 1.008117e-01 1.244875e+00 5.044040e+05
## [6] 1.409726e-03
## [1] 5.000000e+00 4.588019e+02 1.008506e-01 1.244829e+00 5.044032e+05
## [6] 1.492754e-04
## [1] 6.000000e+00 4.587991e+02 1.008520e-01 1.244828e+00 5.044032e+05
## [6] 6.096728e-06
## [1] 7.000000e+00 4.587990e+02 1.008521e-01 1.244827e+00 5.044032e+05
## [6] 2.101742e-07
```

We plot the data together with the fitted values and the least-squares line:

```
yh<-fun(theta=opt$theta,z=data$realdpi)
plot(data$realdpi,data$realcons,xlab="income",ylab="consumption",pch=3)
lines(data$realdpi,yh,col="red",lwd=2)
lines(data$realdpi,reg$fitted.values,col='green',lwd=2,lty=2)
```



We can check that the nonlinear solution is better than the linear one by comparing the residual sum of squares (RSS). For linear regression, it was:

```
print(sum(reg$residuals^2))
```

```
## [1] 1536322
```

With nonlinear regression, we now have:

```
print(sum((data$realcons-yh)^2))
```

```
## [1] 504403.2
```

The RSS has been divided by 3.

Question c

We start by writing a function that computes the LS error as a function of γ, for fixed α and β:

```
g_gamma<-function(gam,alpha,beta,y,z) return(sum((y-alpha-beta*z^gam)^2))
```

and function `LS` that computes the least-squared error:

```
LS<-function(theta,y,z) return(sum((y-fun(theta,z))^2))
```

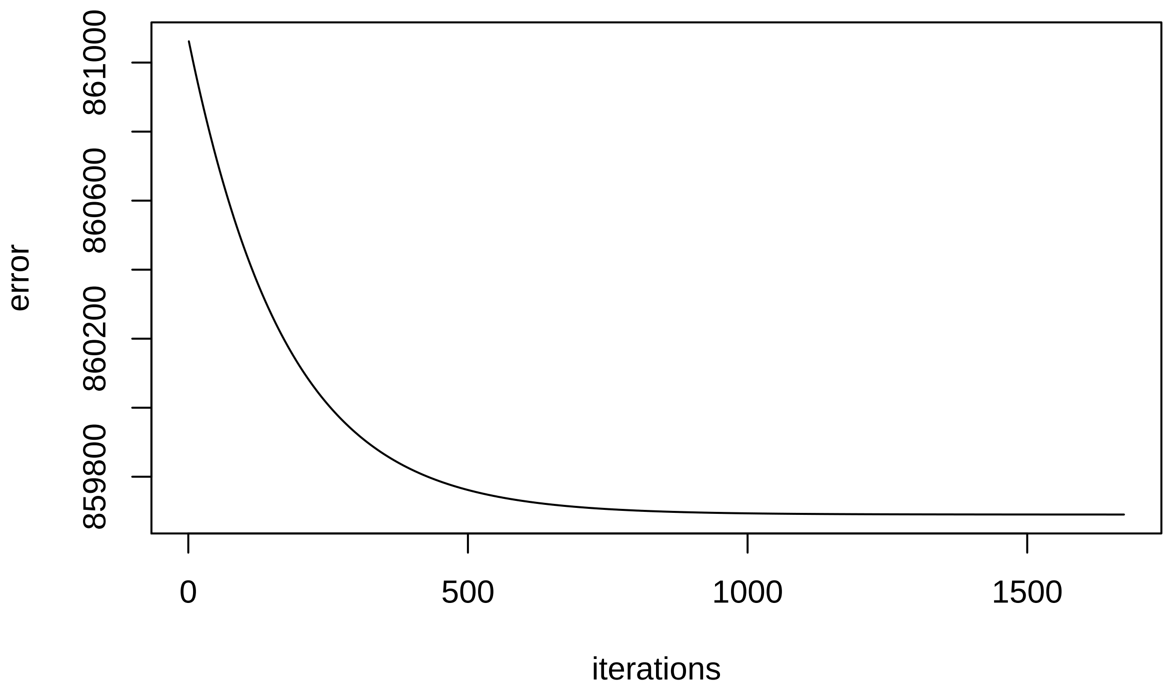
The following code implements the cyclic coordinate descent algorithm: we start with an initial value of γ (e.g., $\gamma = 1.1$), and compute the OLS estimates of α and β for this value of γ. We then optimize γ, for fixed α and β, using the R function `optimize`.

```
z<-data$realdpi   # income
y<-data$realcons # consumption
delta<-1
epsi<-1e-9
theta0<-c(0,0,1.1)
tmax<-10000
```

```
t<-0
g<-rep(0,tmax)
while((delta > epsi)&(t<=tmax)){
  t<-t+1
  z1<-z^theta0[3]
  reg<-lm(y~z1)
  theta<-c(reg$coefficients,theta0[3])
  opt<-optimize(g_gamma,alpha=theta[1],beta=theta[2],y=y,z=z,lower=0.5,upper=2)
  theta[3]<-opt$minimum
  delta<-sum(abs(theta-theta0))/sum(abs(theta0))
  g[t]<-LS(theta,y,z)
  theta0<-theta
}
```

This is a plot of the error vs. the number of iterations:

```
plot(g[2:t],type="l",xlab="iterations",ylab="error")
```



We can see that the cyclic coordinate descent algorithm achieves an error of approximately 8.6e5 after 1500 iterations: it does not perform well on this problem.

Question d

BFGS algorithm:

```
theta0<-c(lm(y~z)$coefficients,1)
opt<-optim(theta0,LS,y=y,z=z,method="BFGS",control=list(trace=3))
```

```
## initial  value 1536321.880788
## iter  10 value 1393750.838469
## iter  20 value 1131754.966699
## iter  30 value 942813.217095
```

```
## iter  40 value 825905.505564
## iter  50 value 732698.970143
## iter  60 value 666000.954976
## iter  70 value 600732.701608
## iter  80 value 568123.011099
## iter  90 value 547450.313226
## iter 100 value 529443.173478
## final  value 529443.173478
## stopped after 100 iterations
```

Nelder-Mead algorithm:

```r
theta0<-c(lm(y~z)$coefficients,1)
opt<-optim(theta0,LS,y=y,z=z,method="Nelder-Mead",control=list(trace=0))
print(opt$value)
```

```
## [1] 651339.3
```

The BFGS algorithm performs as well as the Gauss-Newton, and better than the Nelder-Mead algorithm on this problem.

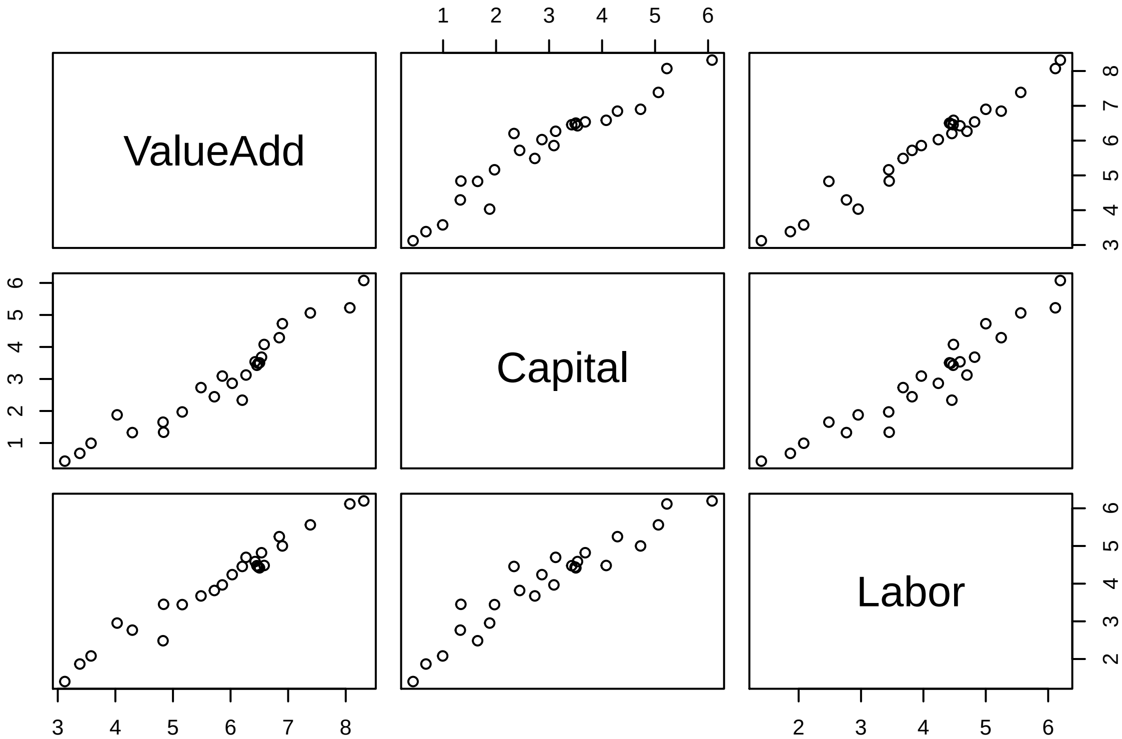
Exercise 4

Question a

```r
transport<-read.table("/Users/Thierry/Documents/R/Data/Compstat/transportation.txt",header=TRUE)
```

Question b

```r
pairs(log(transport))
```



Question c

```
reg <- lm(log(ValueAdd)~log(Capital)+log(Labor),data=transport)
beta0<-coef(reg)
```

Question d

We first need to write a function that computes the log-likelihood:

```
loglik<- function(theta,x,y){
    n<-nrow(x)
    p<-ncol(x)
    beta<-theta[1:(p+1)]
    sigma<-theta[p+2]
    lambda<-theta[p+3]
    epsi<- y-beta[1] - x %*% beta[2:(p+1)]
    loglik<- -n*log(abs(sigma)) + (n/2)*log(2/pi) - 0.5*sum((epsi/sigma)^2) +
             sum(log(pnorm(-epsi*lambda/abs(sigma))))
    return(loglik)
}
```

We then initialize the parameters:

```
beta0<-coef(reg)
sigma0=sqrt(mean(reg$res^2))
lambda0=1
theta0=c(beta0,sigma0,lambda0)
print(theta0)
```

```
##  (Intercept) log(Capital)   log(Labor)                         
##    1.8444157    0.2454281    0.8051830    0.2211117    1.0000000
```

and run the BFGS algorithm with function `optim`:

```
opt<-optim(theta0, loglik, method = "BFGS",control=list(fnscale=-1,trace=2),
           x=cbind(log(transport$Capital),log(transport$Labor)),
           y=log(transport$ValueAdd))
```

```
## initial  value 5.018060 
## iter  10 value -2.469491
## final  value -2.469521 
## converged
```

The maximum of the log-likelihood found is

```
print(opt$value)
```

```
## [1] 2.469521
```

and the corresponding parameter estimates are:

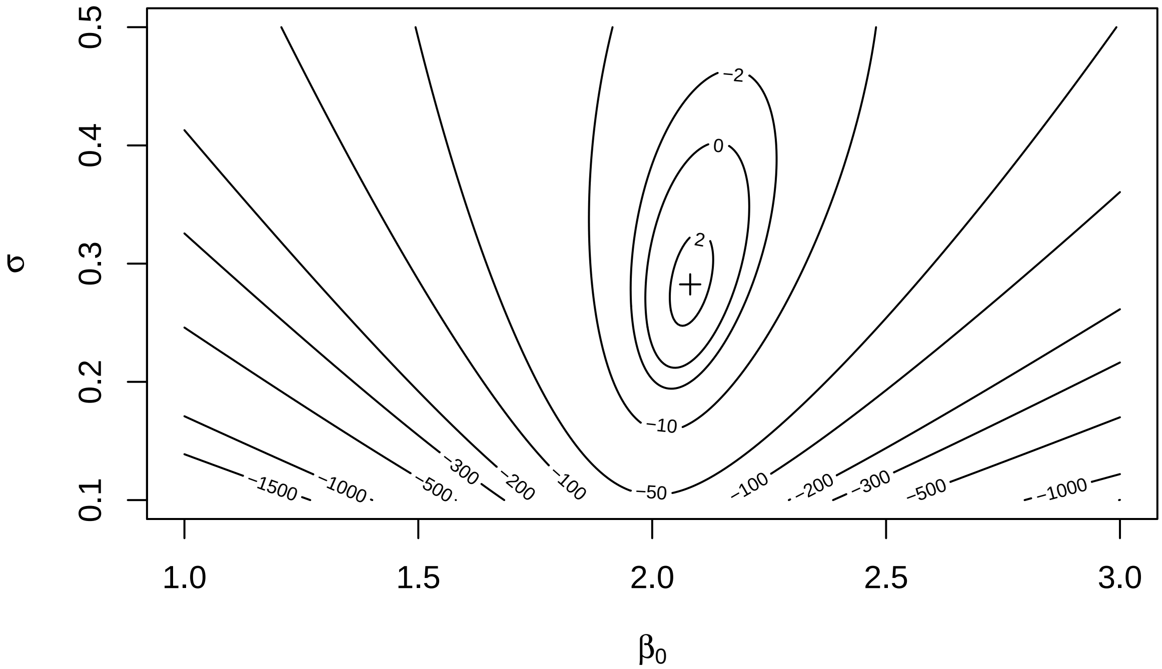
```
print(opt$par)
```

```
##  (Intercept) log(Capital)   log(Labor)                         
##    2.0812043    0.2585327    0.7802561    0.2824470    1.2653270
```

Question e

For example, let us fix all parameters, except β_0 and σ.

```
thetah<-opt$par
xx<-seq(1,3,0.005)
yy<-seq(0.1,0.5,0.0005)
nx=length(xx)
ny=length(yy)
z <- matrix(0,nrow=nx,ncol=ny)
for(i in 1:nx){
    for(j in 1:ny){
        z[i,j]=loglik(c(xx[i],thetah[2:3],yy[j],thetah[5]),x=cbind(log(transport$Capital),log(transport$
    }
}
contour(x=xx,y=yy,z,levels=c(2,0,-2,-10,-50,-100,-200,-300,-500,-1000, -1500))
points(thetah[1],thetah[4],pch=3)
title(xlab=expression(beta[0]),ylab=expression(sigma))
```



We can see that the solution found using `optim` is a local maximum.

Question f

```
theta0=c(3,beta0[2:3],0.5,10)
opt1<-optim(theta0, loglik, method = "BFGS", control=list(fnscale=-1,trace=2),
      x=cbind(log(transport$Capital),log(transport$Labor)),y=log(transport$ValueAdd))
```

```
## initial  value 57.529376
## iter  10 value -0.714958
## iter  20 value -1.029694
## iter  30 value -1.310878
## iter  40 value -1.855346
## final  value -1.872617
## converged
```

```
print(opt1$par)
```

```
##             log(Capital)  log(Labor)
##   2.8500100    0.3035175   0.5978896   0.4391104  42.2383800
```

The solution is very different from the previous one:

```
print(opt1$par)
```

```
##             log(Capital)  log(Labor)
##   2.8500100    0.3035175   0.5978896   0.4391104  42.2383800
```

The corresponding value of the likelihood is lower:

```
print(opt1$value)
```

```
## [1] 1.872617
```

Let us check that it is a local maximum:

```
thetah<-opt1$par
xx<-seq(2.5,3.2,0.005)
yy<-seq(0.2,0.8,0.0005)
nx=length(xx)
ny=length(yy)
z <- matrix(0,nrow=nx,ncol=ny)
for(i in 1:nx){
    for(j in 1:ny){
        z[i,j]=loglik(c(xx[i],thetah[2:3],yy[j],thetah[5]),x=cbind(log(transport$Capital),log(transport$
    }
}
contour(x=xx,y=yy,z,levels=c(1,0,-2,-10,-20,-50,-100,-500,-1000, -2000,-3000,-5000,-7000))
points(thetah[1],thetah[4],pch=20)
title(xlab=expression(beta[0]),ylab=expression(sigma))
```