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

Thierry Denoeux

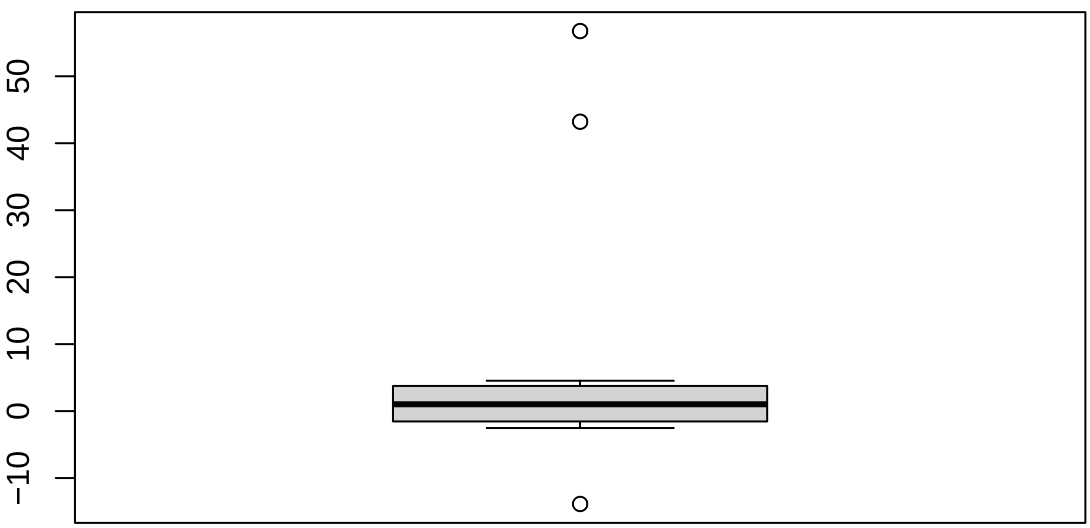
2024-01-23

Exercise 1

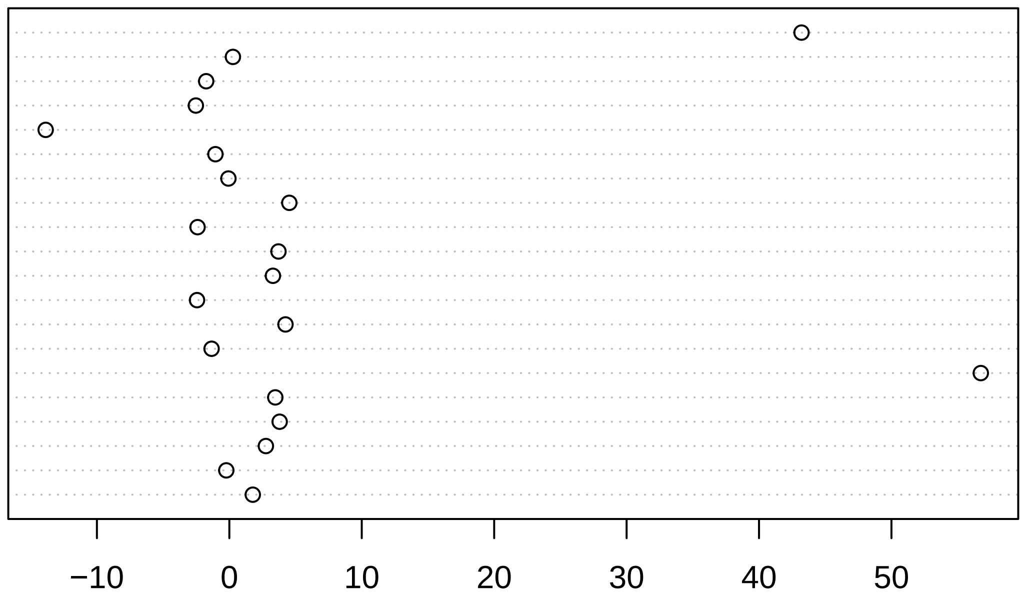
Question a

```
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:

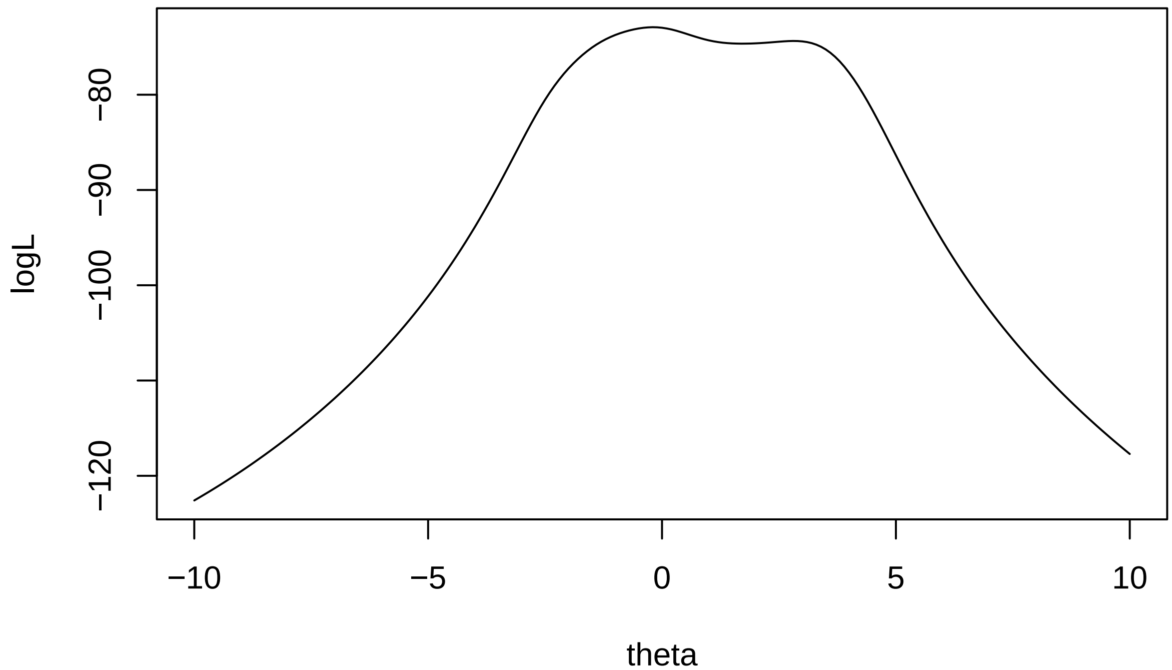
```
theta<- seq(-10,10,0.1)
N<-length(theta)
logL<-rep(0,N)
for(i in 1:N) logL[i]<- loglik(theta[i],x)
```

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:

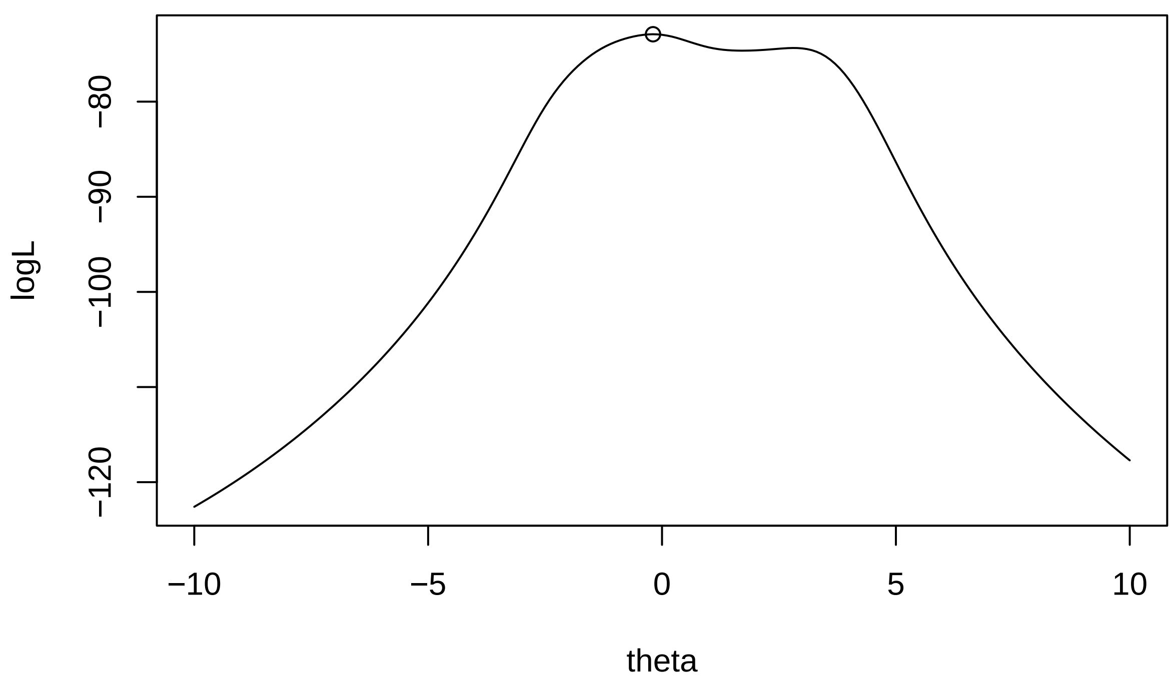
```
bisection <-function(fun,dfun,a,b,epsi,...){
    theta<-(a+b)/2
    delta<-1
    while(delta>epsi){
        theta0<-theta
        if(dfun(a,x)*dfun(theta0,...)<=0) b<-theta0 else a<-theta0
        theta<-(a+b)/2
        delta<-abs(theta-theta0)/(abs(theta0)+epsi)
        print(c(a,b,delta))
    }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

We run it on the data and plot the result:

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

```
## [1] -1.000000  1.000000  0.999999
## [1]     -1      0 500000
## [1] -0.500000  0.000000  0.499999
## [1] -0.250000  0.000000  0.499998
## [1] -0.250000 -0.125000  0.499996
## [1] -0.2500000 -0.1875000  0.1666658
## [1] -0.21875000 -0.18750000  0.07142824
## [1] -0.20312500 -0.18750000  0.03846135
## [1] -0.1953125 -0.1875000  0.0199999
## [1] -0.19531250 -0.19140625  0.01020403
## [1] -0.193359375 -0.191406250  0.005050479
## [1] -0.192382812 -0.191406250  0.002538058
## [1] -0.192382812 -0.191894531  0.001272258
## [1] -0.1923828125 -0.1921386719  0.0006353207
## [1] -0.1923828125 -0.1922607422  0.0003174587
## [1] -0.192321777 -0.192260742  0.000158679
## [1] -1.922913e-01 -1.922607e-01  7.935207e-05
## [1] -1.922913e-01 -1.922760e-01  3.967918e-05
## [1] -0.1922912598 -0.1922836304  0.0000198388
## [1] -1.922874e-01 -1.922836e-01  9.919206e-06
## [1] -1.922874e-01 -1.922855e-01  4.959652e-06
## [1] -1.922874e-01 -1.922865e-01  2.479814e-06
## [1] -1.922870e-01 -1.922865e-01  1.239904e-06
## [1] -1.922867e-01 -1.922865e-01  6.199527e-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:

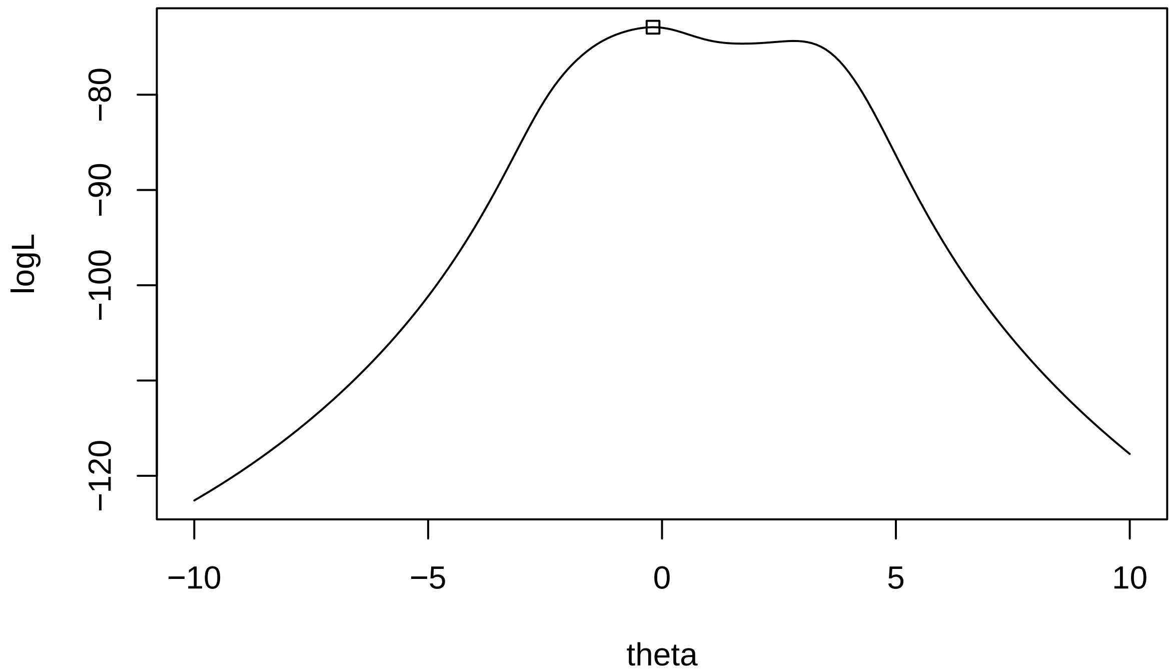
```
newton <-function(fun,dfun,d2fun,theta0,epsi,tmax,...){
    delta=1
    t<-0
    while((delta>epsi)&(t<=tmax)){
      t<-t+1
        theta<-theta0-dfun(theta0,...)/d2fun(theta0,...)
        delta<-abs(theta-theta0)/(abs(theta0)+epsi)
        obj<-fun(theta,...)
        print(c(t,theta,obj,delta))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

We run it on our data and plot the results:

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

```
## [1]  1.000000e+00 -1.963366e-01 -7.291584e+01  1.963366e+05
## [1]   2.00000000  -0.19228252 -72.91581962   0.02064847
## [1]  3.000000e+00 -1.922866e-01 -7.291582e+01  2.126305e-05
## [1]  4.000000e+00 -1.922866e-01 -7.291582e+01  2.109083e-11
```

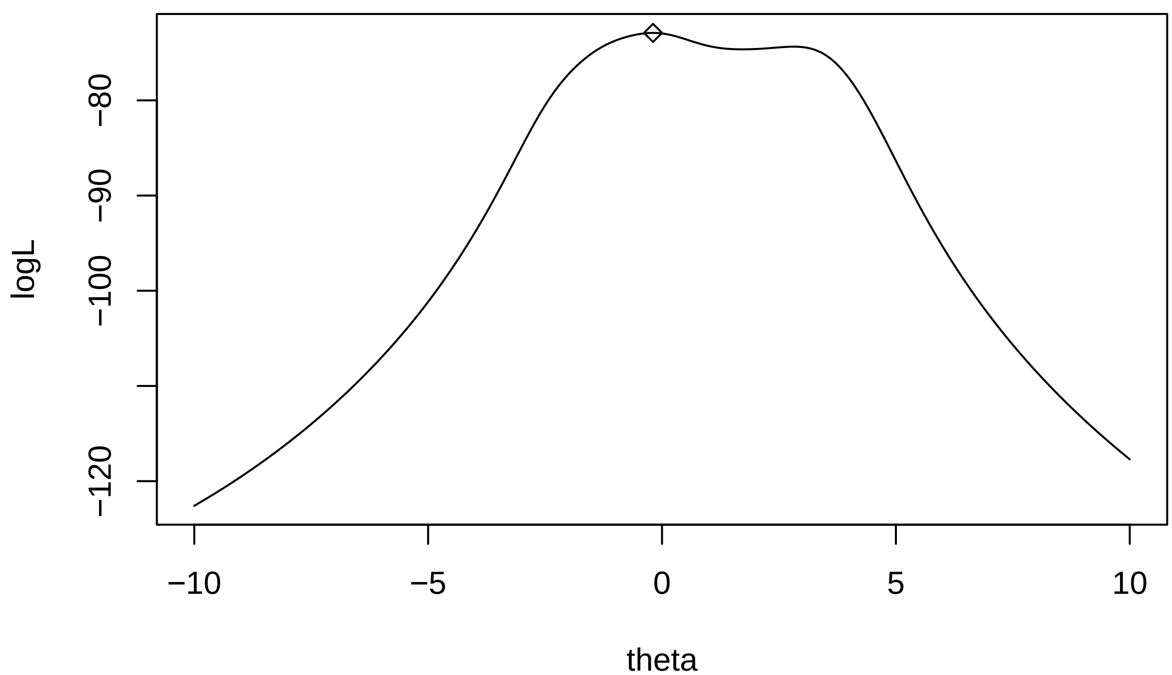
```
plot(theta,logL,type="l")
points(opt$optimum,opt$objective,pch=22)
```



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))
}
```

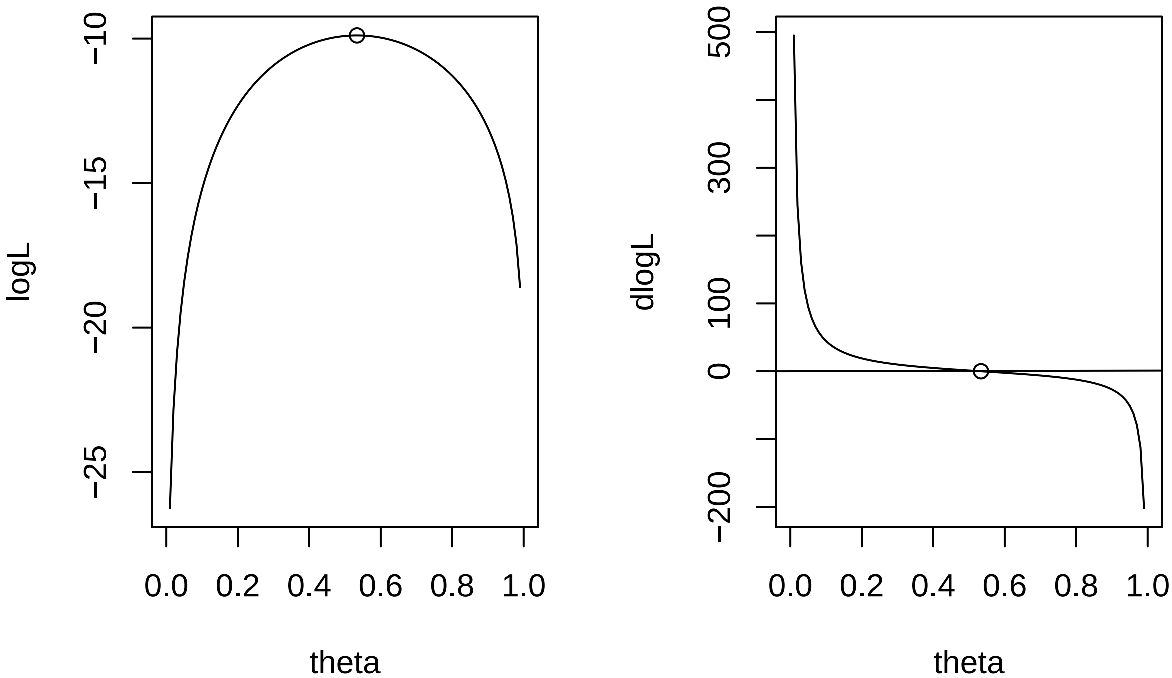
We use function `newton` from Exercise 1:

```
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   0.1871685
## [1]  2.0000000  0.5444798 -9.8929451  0.1626798
## [1]  3.00000000  0.53359344 -9.89093316  0.01999409
## [1]  4.000000e+00  5.335892e-01 -9.890933e+00  7.890570e-06
## [1]  5.000000e+00  5.335892e-01 -9.890933e+00  1.474360e-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

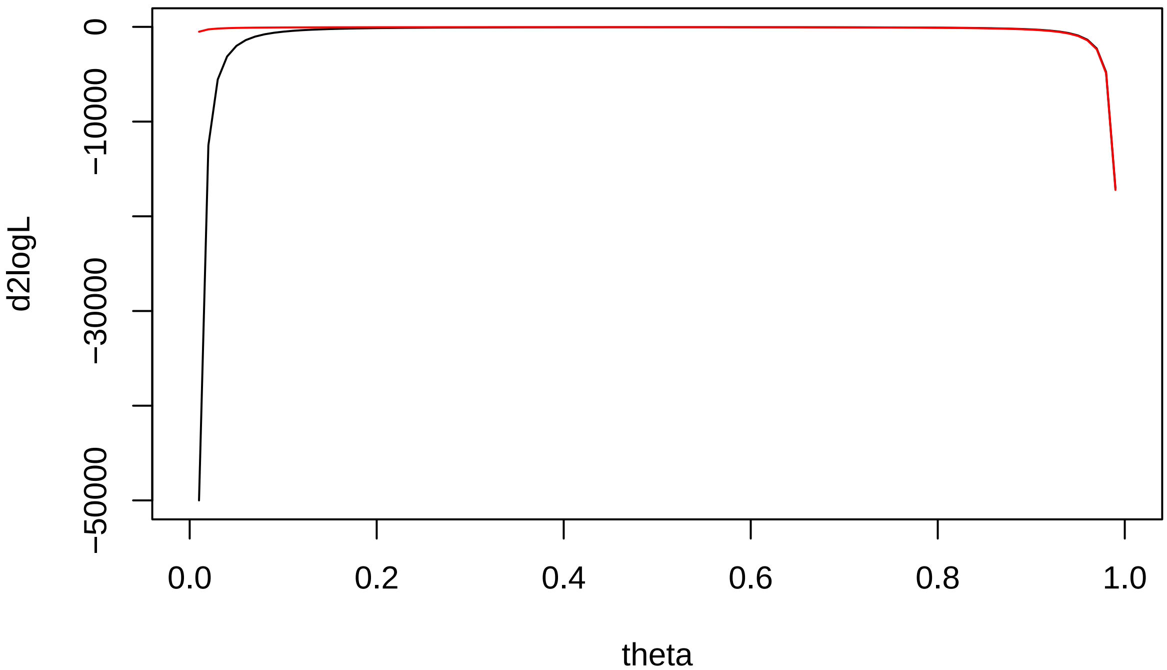
$$\mathbb{E}_\theta[X] = \frac{-\theta}{(1-\theta)\log(1-\theta)}.$$

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")
```



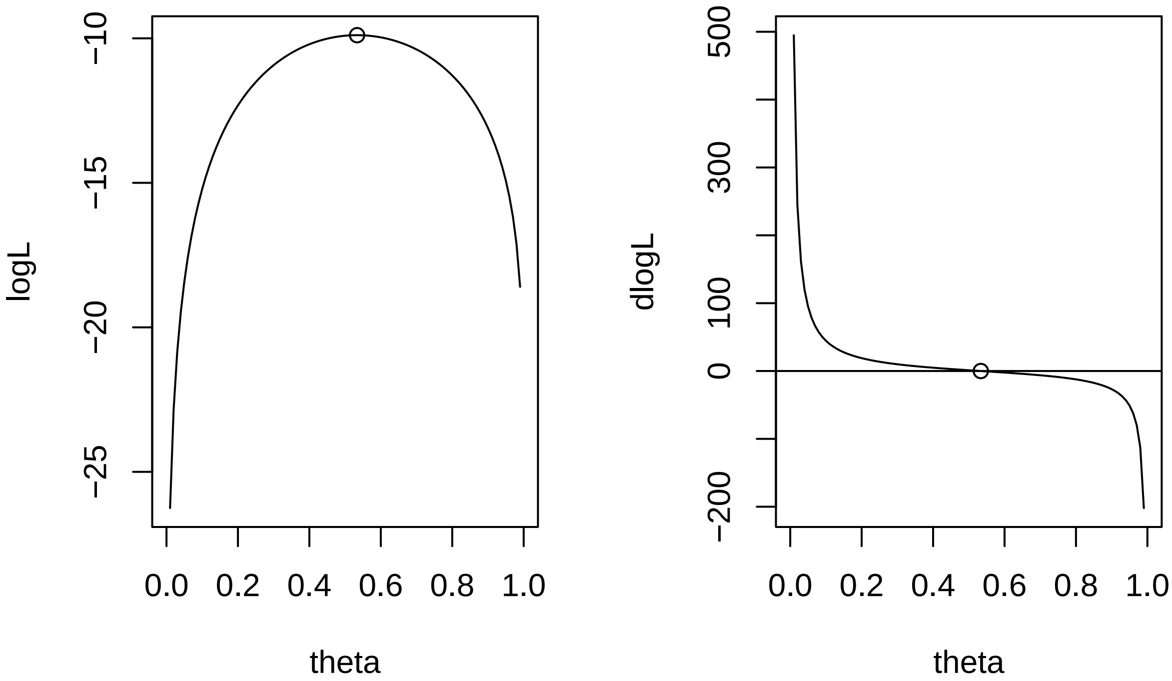
To use the Fisher scoring method, we can use function `newton` and pass minus the Fisher information instead of the second derivative as argument:

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

```
## [1]   1.0000000   0.6738722 -10.2362220   0.1576596
## [1]  2.0000000  0.5709805 -9.9146869  0.1526870
## [1]  3.00000000  0.53617188 -9.89104631  0.06096278
## [1]  4.000000000  0.533601446 -9.890933165  0.004794031
## [1]  5.000000e+00  5.335892e-01 -9.890933e+00  2.288544e-05
## [1]  6.000000e+00  5.335892e-01 -9.890933e+00  5.113807e-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

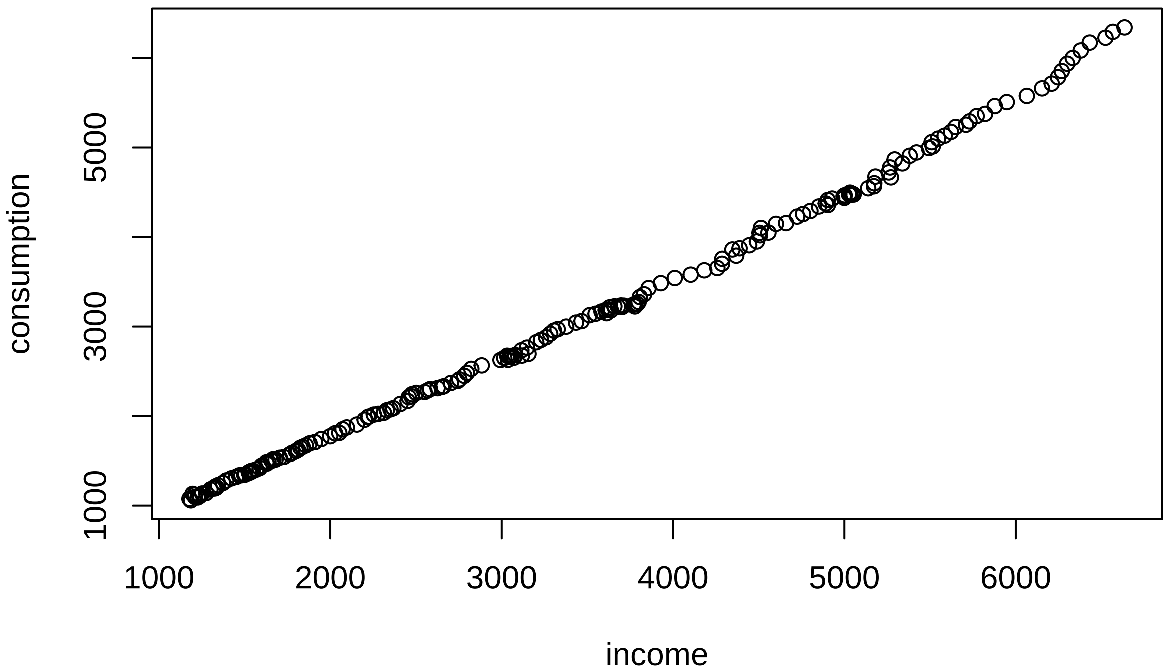
Question a

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")
```



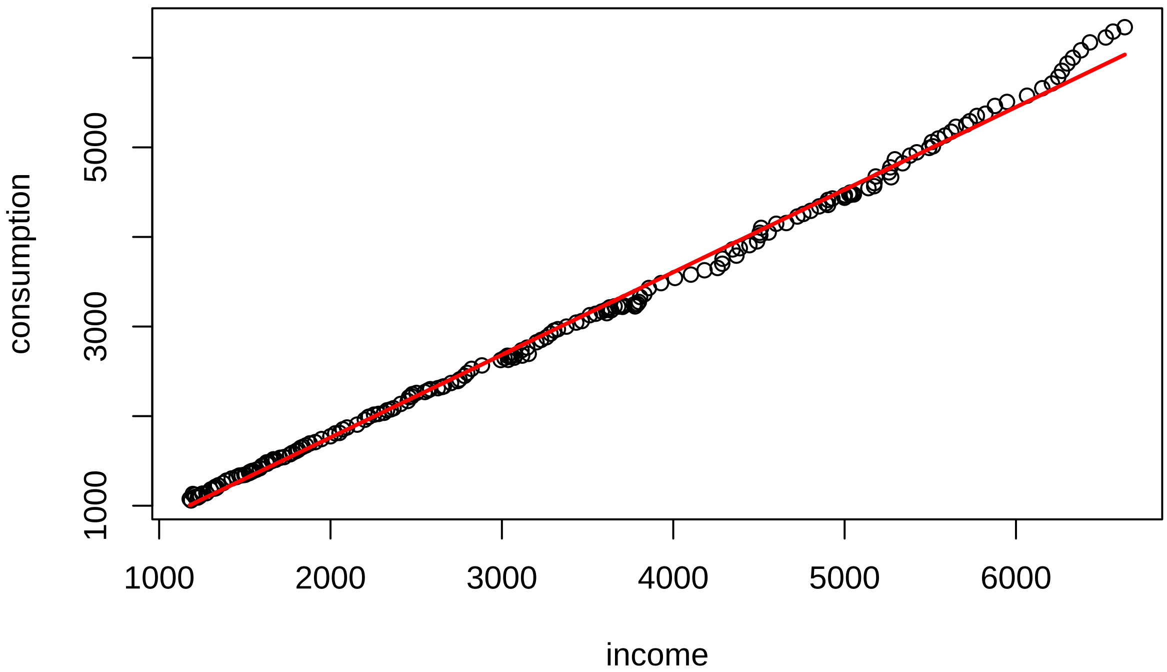
The relationship between consumption and income seems approximately linear. We then estimate α and β using linear regression and display a summary of the result:

```
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
```

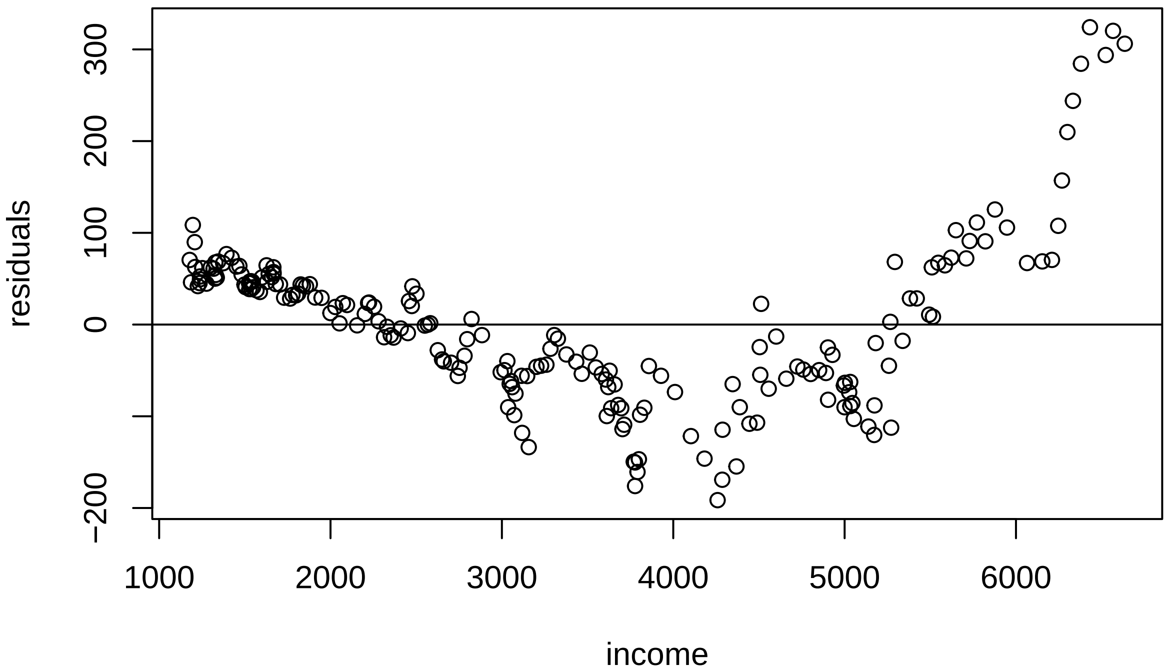
A plot of the least-squares line together with the data seems to indicate a good fit:

```
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

To apply the BFGS algorithm implemented in function `optim`, we first write a function that compute the residual sum-of-squares criterion:

```
RSS<-function(theta,y,z){
  yhat <- theta[1]+theta[2]*z^theta[3]
  return(sum((y-yhat)^2))
}
```

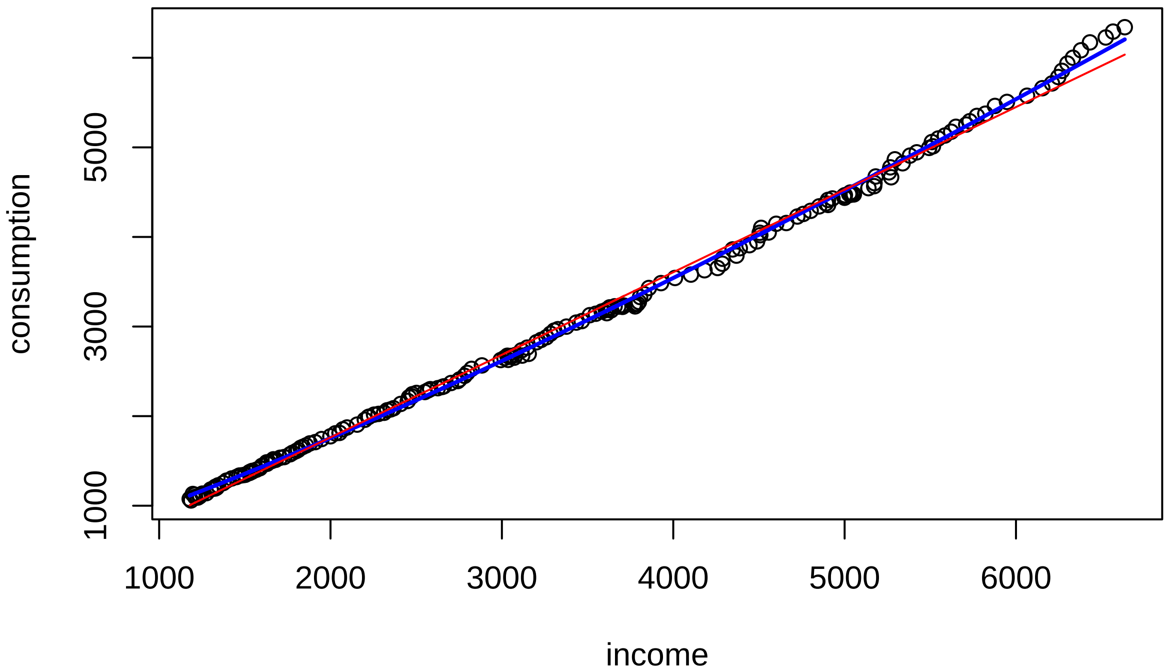
We can then run function `optim`, using the linear regression estimates of coefficients α and β with $\gamma = 1$ as initial estimates:

```
theta0 <- c(reg$coefficients,1)
opt <- optim(theta0,RSS,y=data$realcons,z=data$realdpi,method="BFGS",
             control=list(trace=3,maxit=1000))
```

```
## initial  value 1536321.880788
## iter  10 value 1393750.823599
## iter  20 value 1131753.022894
## iter  30 value 942822.914783
## iter  40 value 824997.561004
## iter  50 value 731509.516342
## iter  60 value 664774.066417
## iter  70 value 600208.014377
## iter  80 value 567557.604705
## iter  90 value 541915.432076
## iter 100 value 530739.243338
## iter 110 value 521066.188498
## final  value 520866.618992
## converged
```

Plotting the results:

```
theta <- opt$par
yhat<-theta[1]+ theta[2]*data$realdpi^theta[3]
plot(data$realdpi,data$realcons,xlab="income",ylab="consumption")
lines(data$realdpi,yhat,col="blue",lwd=2)
lines(data$realdpi,reg$fitted.values,col="red")
```



We can check that the nonlinear solution is better than the linear one by comparing the RSS. For linear regression, it was:

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

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

With nonlinear regression, we now have:

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

```
## [1] 520866.6
```

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))
```

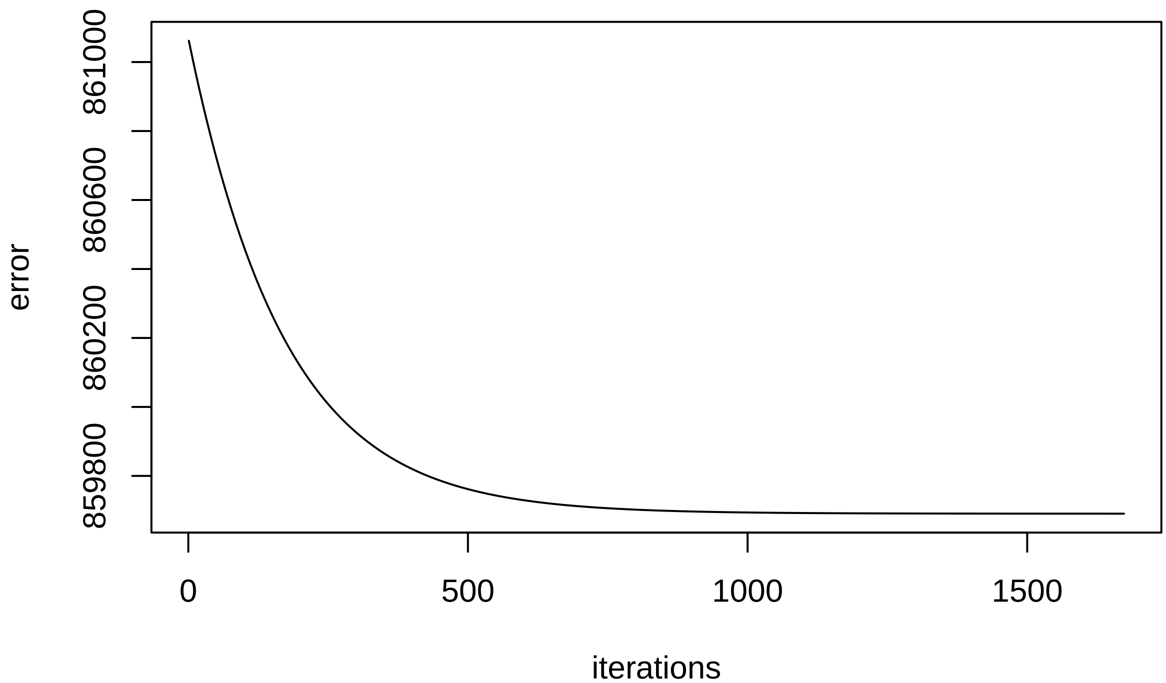
The following code implements the cyclic coordinate descent algorithm: we start with an initial value of γ (e.g., $\gamma = 1.1$), and compute the OLS estimates of α and β for this value of γ. We then optimize γ, for fixed α and β, using the R function `optimize`.

```
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]<-RSS(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

We will now implement an optimization strategy that exploits the particular form of the problem: the Gauss-Newtion algorithm.

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

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