Problem Set 8
STAT243: Statistical Computing
University of California, Berkeley

Thibault Doutre, Student ID 26980469

December, 2015

1 Bootstrap regression

We wish to compare the OLS estimator with one more robust method. Given X and Y such that $Y = X\beta + \epsilon$ with $\epsilon \sim N(0, 1)$, the OLS estimator is obtained by minimizing the square loss and the robust method used here minimizes the absolute loss.
When treating the estimate β as random variables, the Bootstrap can be used to estimate the confidence intervals.

Let B be the number of the bootstrapped samples. Let $X \in M_{n,p}(R)$ be the data and $Y \in R^n$ be the observations. The bootstrapped estimation of the regression parameter β consists of the following:

For $b \in \{0..B-1\}$ do:

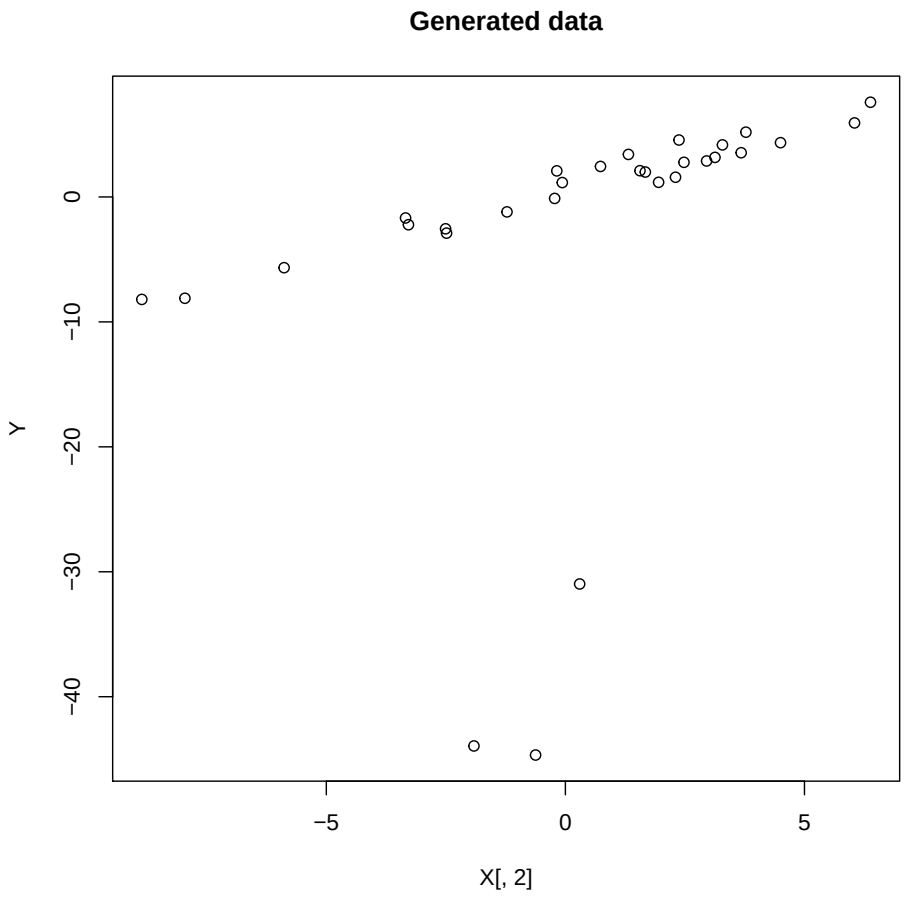
- Create $X_b \in M_{n,p}(R)$ and $Y_b \in R^n$ such that the lines of X_b are sampled with replacement from the lines of X.
- Estimate $\hat{\beta}_b$ with the regression method used.

The bootstrapped mean estimator of β, say β^* is the mean of the $\hat{\beta}_b$ for the OLS regression and the median of $\hat{\beta}_b$ for the robust regression. In order to have the confidence interval of the parameter of interest, one needs to compute the standard deviation of the parameters $\hat{\beta}_b$ now treated as random variables, say $SE \in R^p$. Then the confidence interval is then given by:

$$\left[\beta^* - SE * q_{1-\alpha/2}^{N(0,1)}, \beta^* + SE * q_{1-\alpha/2}^{N(0,1)}\right] \tag{1}$$

I chose to use robust LM from lmrob function in R. First, generate data and add some outliers.

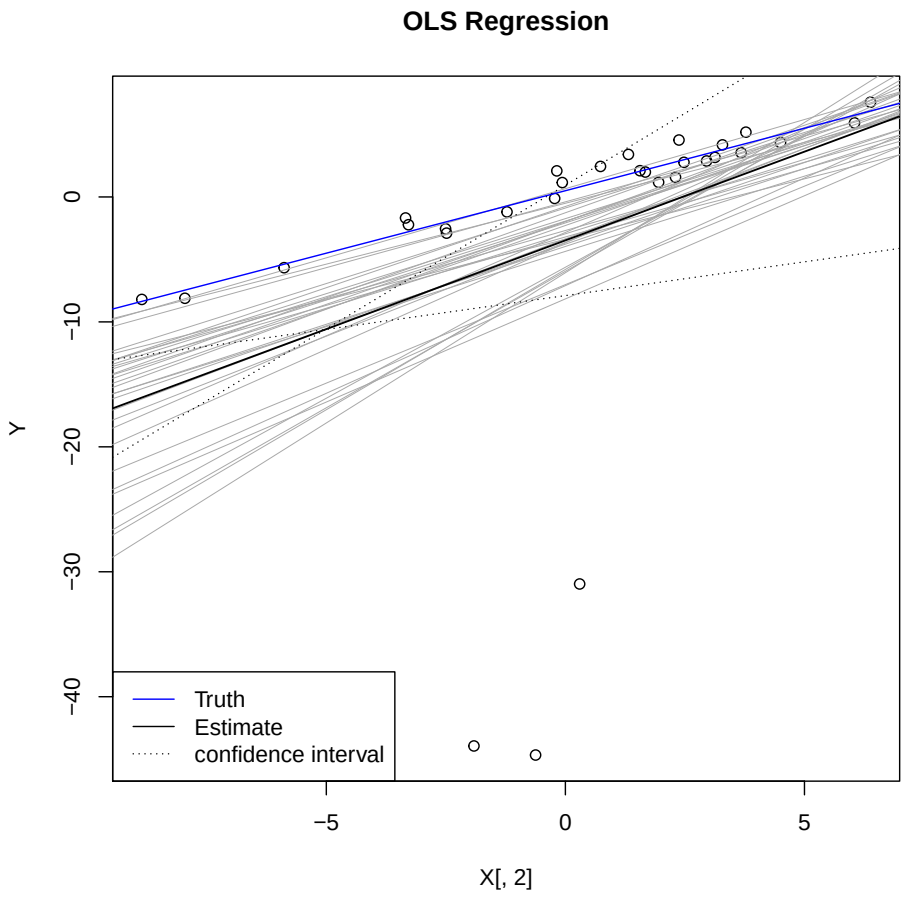
```
set.seed(1)
library(robust)

## Loading required package:  fit.models
## Loading required package:  lattice
## Loading required package:  MASS
## Loading required package:  robustbase
## Loading required package:  rrcov
## Scalable Robust Estimators with High Breakdown Point (version
1.3-8)

options(warn=-1)

# Generate data
n=30
p=4
beta = matrix(c(.5,1,0,0),ncol=1)
X = cbind(1,rnorm(n,sd=4),rnorm(n,sd=.2),rnorm(n,sd=.2))
Y = X%*%beta+matrix(rnorm(n),ncol=1) #Good data
Y[sample(1:n,n/10)] = Y[sample(1:n,n/10)] +
  rnorm(0,40,n=n/10) #Bad data 10%
plot(X[,2],Y,main="Generated data")
```



Then, I compute the OLS estimation, choosing $B = 1000$.

```
################## OLS ######################
plot(X[,2],Y,main="OLS Regression")

# Bootsrap sample
B=1000
r=n
OLSbetastar=matrix(nrow=B,ncol=p)
for (b in 1:B){
  s=sample(1:n,r,replace=T)
  Xstar=X[s,]
  Ystar=Y[s]
```

```r
  OLSbetastar[b,]=lm(Ystar~Xstar[,-1])$coefficients
}

# Bootsrapped estimates
OLSbetastar_hat=apply(OLSbetastar,2,mean)
sum(abs(Y-X%*%OLSbetastar_hat)) # Absolute error

## [1] 214.9669

for (row in 1:n){
  abline(OLSbetastar[row,1:2],col="darkgray",lwd=0.3)
}
abline(OLSbetastar_hat[1:2],lwd=1.3)

# Bootstrapped confint 95%
SE=apply(OLSbetastar,2,sd)
borne_sup=OLSbetastar_hat+qnorm(1-.05/2)*SE
borne_inf=OLSbetastar_hat-qnorm(1-.05/2)*SE
abline(borne_sup[1:2],lty=3)
abline(borne_inf[1:2],lty=3)

# True beta
abline(beta[1:2],col="blue")

legend("bottomleft",c("Truth","Estimate","confidence interval"),
       lty=c(1,1,3),col=c("blue","black","black"))
```

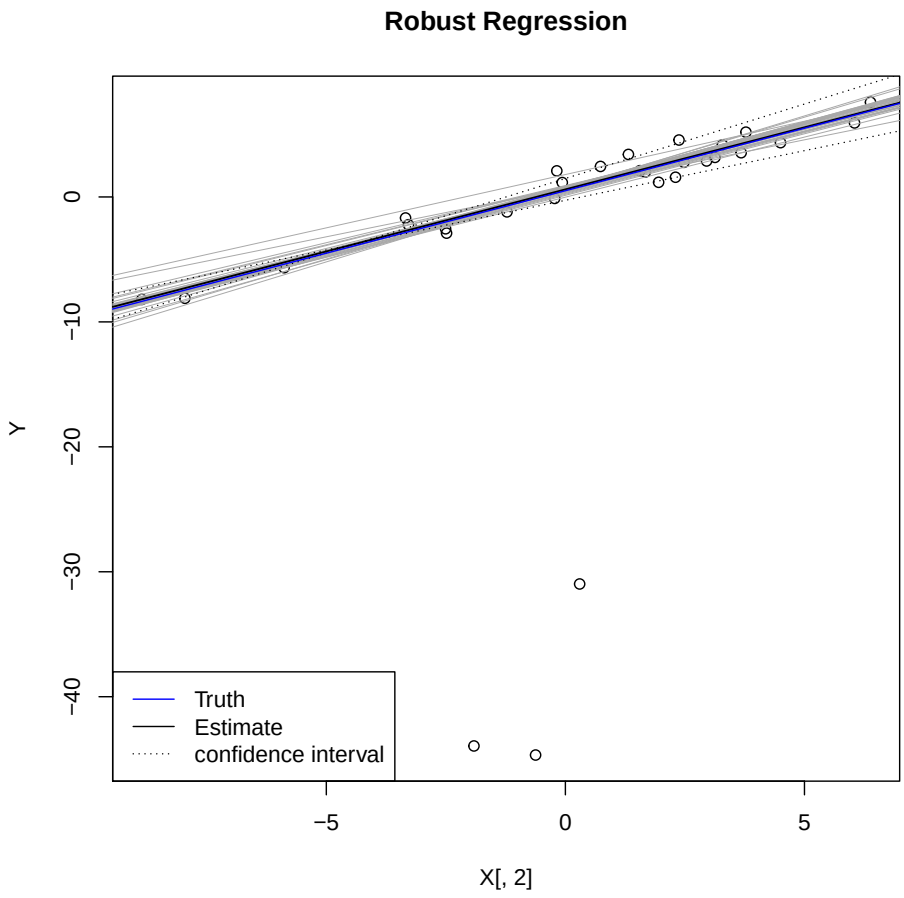



```
# Coverage prediction
mean(apply(OLSbetastar,1,function(row) Reduce('*',row>borne_inf && row<borne_sup)))

## [1] 0.962
```

We can see that the confidence interval is indeed 0.95. To compare the robust method with standard OLS, I do the exact same steps than before with the robust method.

```
################## ROBUST ######################
plot(X[,2],Y,main="Robust Regression")
```

```r
# Bootsrap sample
B=1000
r=n
betastar=matrix(nrow=B,ncol=p)
for (b in 1:B){
  s=sample(1:n,r,replace=T)
  Xstar=X[s,]
  Ystar=Y[s]
  betastar[b,]=lmrob(Ystar~Xstar[,-1],tau=.5)$coefficients
}

# Bootsrapped estimates
betastar_hat=apply(betastar,2,mean)
sum(abs(Y-X%*%betastar_hat)) # Absolute error

## [1] 139.0308

for (row in 1:n){
  abline(betastar[row,1:2],col="darkgray",lwd=0.3)
}
abline(betastar_hat[1:2],lwd=1.3)

# Bootstrapped confint 95%
SE=apply(betastar,2,sd)
borne_sup=betastar_hat+qnorm(1-.05/2)*SE
borne_inf=betastar_hat-qnorm(1-.05/2)*SE
abline(borne_sup[1:2],lty=3)
abline(borne_inf[1:2],lty=3)

# True beta
abline(beta[1:2],col="blue")

legend("bottomleft",c("Truth","Estimate","confidence interval"),
       lty=c(1,1,3),col=c("blue","black","black"))
```




```
# Coverage prediction
mean(apply(betastar,1,function(row) Reduce('*',row>borne_inf && row<borne_sup)))

## [1] 0.985
```

And I finally report the absolute errors:

```
sum(abs(Y-X%*%betastar_hat)) # LMROB_boot Absolute error

## [1] 139.0308

sum(abs(Y-X%*%OLSbetastar_hat)) # OLS Absolute error

## [1] 214.9669
```

We note that theOLS estimate is worse than the robust one because of these these outliers.

2 Monte Carlo

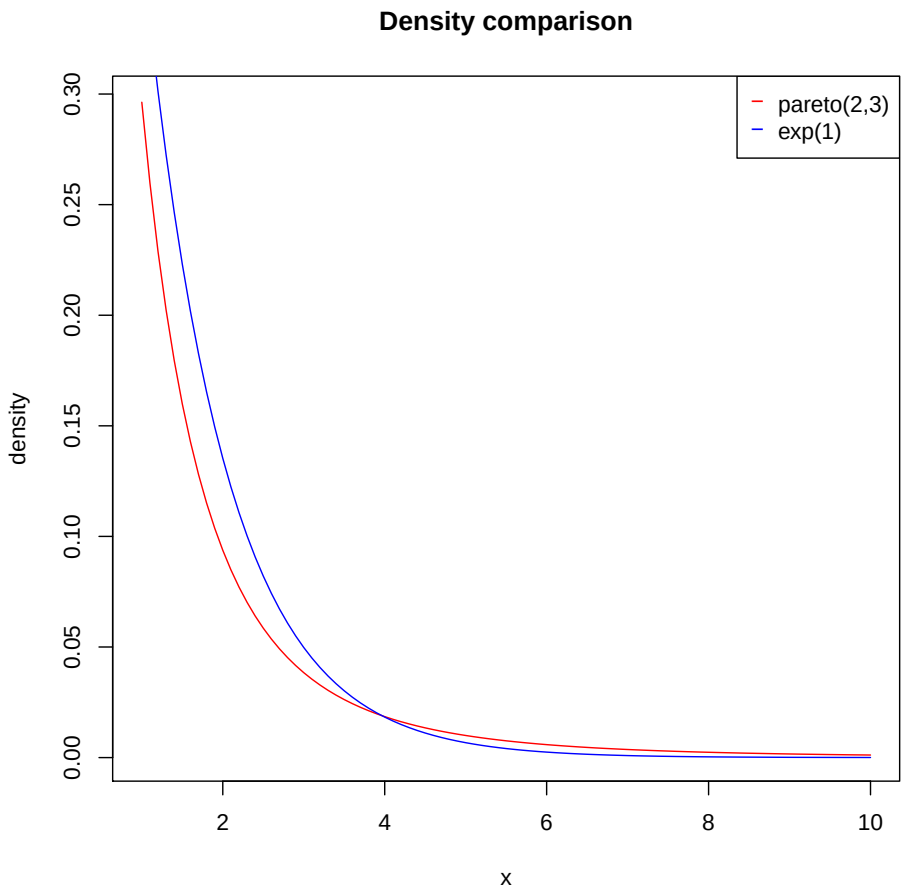
The tail of the Pareto decay more slowly than that of an exponential distribution. Indeed, when taking the ratio of the tails of the distributions, we have:

$$\frac{e^{-\lambda x}}{\alpha^{\beta}/x^{\beta}} = \frac{x^{\beta}e^{-\lambda x}}{\alpha^{\beta}} \xrightarrow[x\to\infty]{} 0 \tag{2}$$

We can confirm this mathematical statement when plotting the curves:

```
library(actuar)

##
## Attaching package:  'actuar'
##
## The following object is masked from 'package:grDevices':
##
##     cm
```

```
# Plot
x=seq(1,10,by=0.1)
plot(x,dpareto(x,shape=3,scale=2),type="l",col="red",ylab=
       "density",main="Density comparison")
lines(x,dexp(x),col="blue")
legend("topright",c("pareto(2,3)","exp(1)"),col=c("red","blue")
       ,pch="--")
```



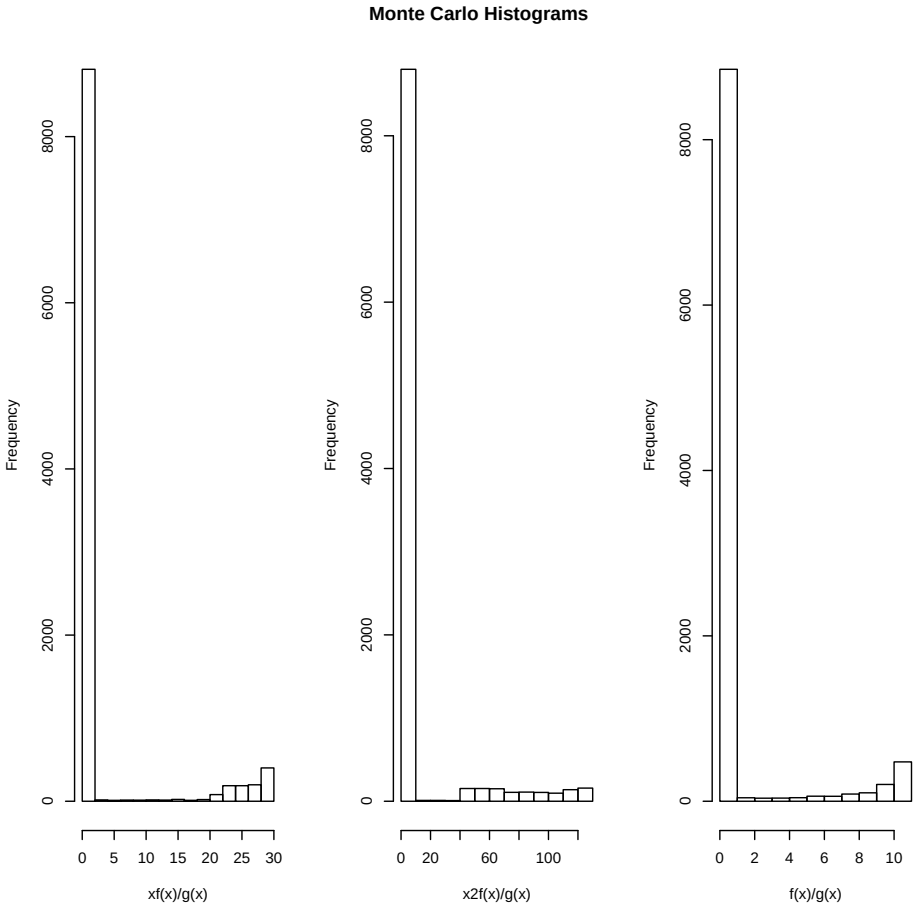
2.1 Sampling density with heavier tail

If we sample the shifted exponential distribution with the Pareto distribution, we have:

```
# Moment estimation
f = function(x){
  dexp(x-2)
}
g = function(x){
  dpareto(x,shape=3,scale=2)
}
```

```r
w=function(x){
  f(x)/g(x)
}
h1=function(x){
  x*w(x)
}
h2=function(x){
  (x^2)*w(x)
}

m=10000
sample=rpareto(m,shape=3,scale=2)

#E(X)
mean(h1(sample))

## [1] 2.933696

#E(X2)
mean(h2(sample))

## [1] 9.89575
```

```r
#plot
plot.new()
par(mfrow=c(1,3))
hist(h1(sample),xlab="xf(x)/g(x)",main="")
hist(h2(sample),xlab="x2f(x)/g(x)",main="")
hist(w(sample),xlab="f(x)/g(x)",main="")
title("Monte Carlo Histograms",outer=T,line=-3)
```



We can see that there is not any extreme weights values, according to the range of the histogram or by computing the maximum of the weights:

```
#maxweight
max(w(sample))

## [1] 10.66666
```

2.2 Sampling density with lighter tail

Here, we can see that we do have extreme weights in some cases. When sampling with the shifted exponential distribution with the Pareto, we have:

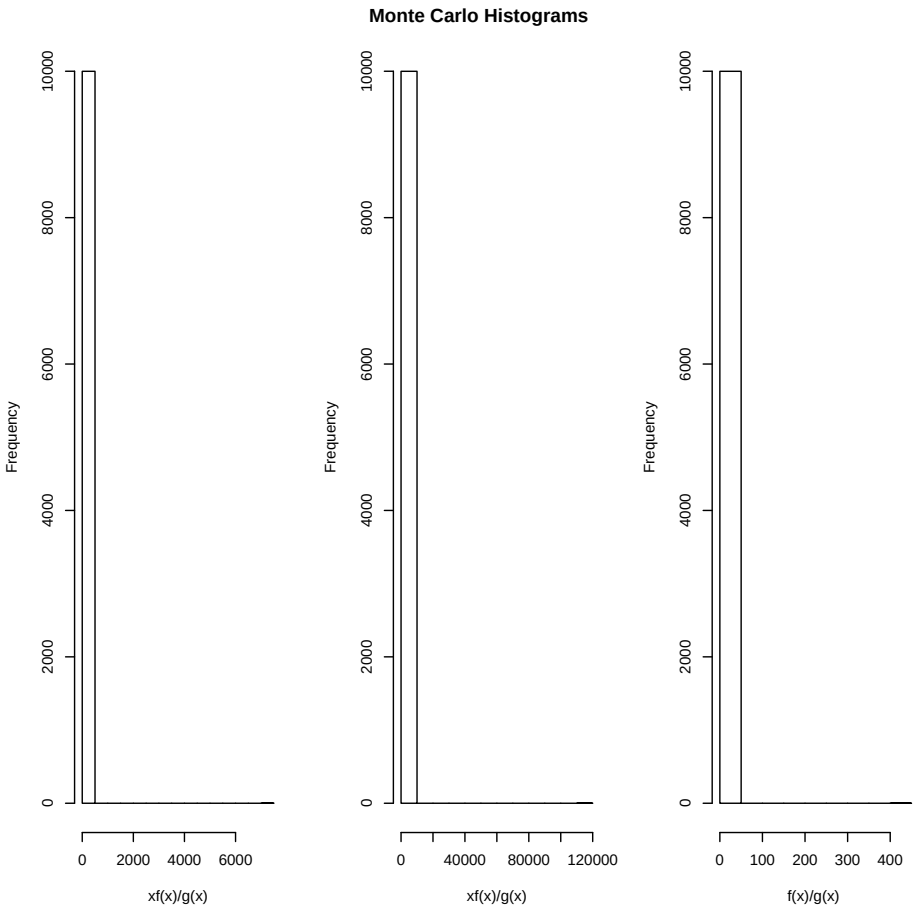
```r
# Exchange f and g
f = function(x){
  dpareto(x,shape=3,scale=2)
}
g = function(x){
  dexp(x-2)
}
w=function(x){
  f(x)/g(x)
}
h1=function(x){
  x*w(x)
}
h2=function(x){
  (x^2)*w(x)
}
m=10000
set.seed(6)
sample=rexp(m)+2
#E(X)
mean(h1(sample))
```

```
## [1] 1.142526
```

```r
#E(X2)
mean(h2(sample))
```

```
## [1] 13.76207
```

```r
#plot
plot.new()
par(mfrow=c(1,3))
hist(h1(sample),xlab="xf(x)/g(x)",main="")
hist(h2(sample),xlab="xf(x)/g(x)",main="")
hist( w(sample),xlab="f(x)/g(x) ",main="")
title("Monte Carlo Histograms",outer=T,line=-3)
```




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

```
#maxweight
max(w(sample))

## [1] 434.0526
```

It is actually important to have the ratio of the density of interest over the sampling density upper bounded in absolute value.

3 EM

The probit regression can be rewritten as:

$$\begin{aligned} Z_i &= X_i^T \beta + \epsilon \qquad\qquad \epsilon \sim N(0,1) \\ Y_i &= I(Z_i > 0) \end{aligned} \tag{3}$$

The expected complete log-likelihood is:

$$Q(\beta) = -\frac{1}{2} E[(Z - X\beta)^T (Z - X\beta) | Y, \beta] + cst \tag{4}$$

Assuming that we can interchange the derivative and the integral operators we have:

$$\frac{dQ}{d\beta}(\beta) = E[X^T Z - X^T X \beta | Y, \beta] \tag{5}$$

Setting this derivative to zero:

$$\hat{\beta} = (X^T X)^{-1} X^T E[Z | Y, \beta] \tag{6}$$

$Z_i|Y$ has as a truncated normal distribution with $\tau = 0$. Therefore:

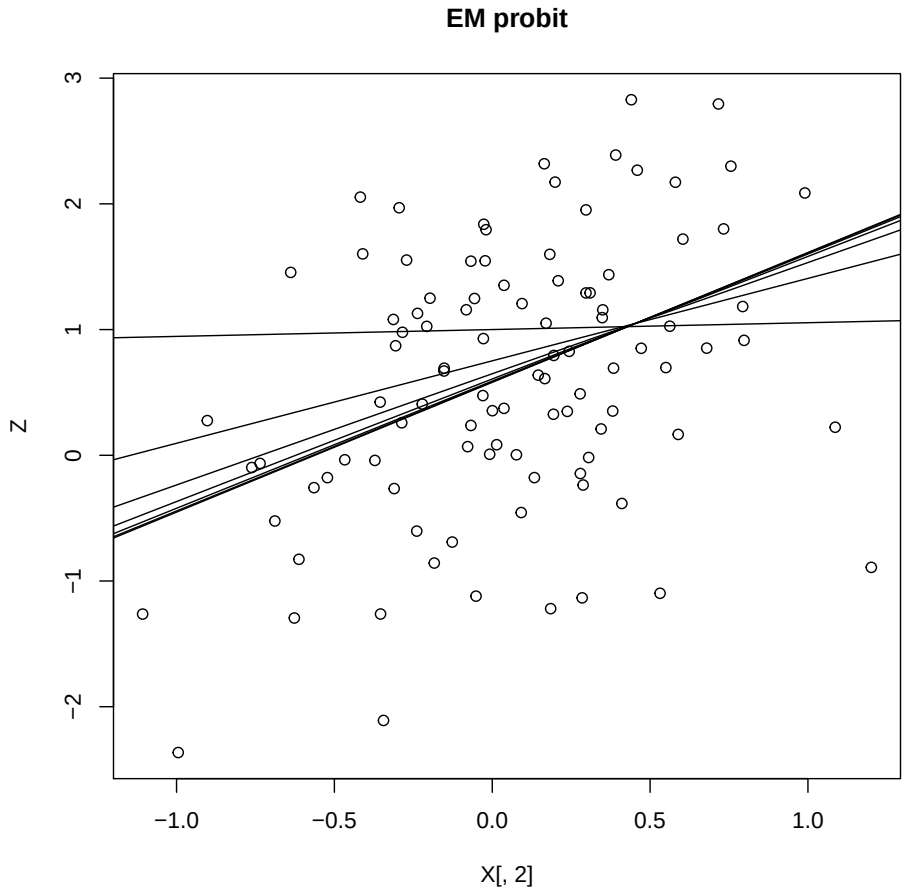
$$\begin{aligned} E[Z_i | Y_i = 1, \beta] &= E[Z_i | Z_i > 0, \beta] \\ &= X_i^T \beta + \frac{\phi(-X_i^T \beta)}{1 - \Phi(-X_i^T \beta)} \\ E[Z_i | Y_i = 0, \beta] &= X_i^T \beta - \frac{\phi(-X_i^T \beta)}{\Phi(-X_i^T \beta)} \end{aligned} \tag{7}$$

Therefore, the EM algorithm can be written as:

- Initialize $\hat{\beta}^{(0)}$.
- Until convergence:
  - Compute $E[Z|Y, \hat{\beta}^{(t)}]$ with (5).
  - Update $\hat{\beta}^{(t+1)}$ with (4).

```
set.seed(1)
# Auxiliaries functions
Estep=function(x,y,beta){
  z=matrix(0,ncol=1,nrow=length(y))
  for (i in 1:length(y)){
    xi=x[i,]
    xib=crossprod(xi,beta)
    if (y[i]==0){
      z[i,]=xib-dnorm(xib)/(pnorm(-xib))
    }
    else{
      z[i,]=xib+dnorm(xib)/(1-pnorm(-xib))
    }
  }
  return(z)
}
Mstep =function(X,Z){
  as.matrix(lm(Z~X[,-1])$coefficients,ncol=1)
}

# EM
em = function(X,beta0,eps,plot=F){
  beta_old=beta0
  beta=beta0
  nit=0
  while (nit==0 | sum((beta_old-beta)^2)/sum((beta_old)^2)>eps){
    beta_old=beta
    if(plot)
      abline(beta[1:2])
    Z=Estep(X,Y,beta)
    beta=Mstep(X,Z)
    Y=as.numeric(Z>0)
    nit=nit+1
  }
  return(list(beta=beta,nit=nit))
}

# Generate data
beta = matrix(c(.5,1,0,0),ncol=1)
```

```r
X = cbind(1,rnorm(100,sd=.5),rnorm(100,sd=2),rnorm(100,sd=2))
Z = X%*%beta+matrix(rnorm(100),ncol=1)
Y=as.numeric(Z>0) # Observed data
plot(X[,2],Z,main="EM probit")

# Initialize Beta
beta0=apply(X,2,mean)
# Run EM
em(X,beta0,1e-5,plot=T)
```



```
## $beta
##                    [,1]
## (Intercept) 0.57934758
## X[, -1]1    1.03509061
## X[, -1]2    0.04487137
## X[, -1]3    0.01732625
##
## $nit
## [1] 7
```

If Z is observed, we can directly maximize the likelihood of the data, i.e. minimize the loss which is:

$$L(\beta) = (Z - X\beta)^T(Z - X\beta) \tag{8}$$

In order to use the BGFS algorithm, we need the gradient of this loss function which is:

$$\frac{dL}{d\beta}(\beta) = -X^T Z + X^T X\beta \tag{9}$$

The standard errors of the estimate $\hat{\beta}$ are given by the square root of the diagonal terms of the following matrix:

$$Est.Var[\hat{\beta}] = \frac{\hat{\epsilon}^T\hat{\epsilon}}{n-p}(X^T X)^{-1} \tag{10}$$

```
# data
beta = matrix(c(.5,1,0,0),ncol=1)
X = cbind(1,rnorm(100,sd=.5),rnorm(100,sd=2),rnorm(100,sd=2))
Z = X%*%beta+matrix(rnorm(100),ncol=1)
Y=Z # Observed data

# BFGS
loss = function(beta){
  r=Y-X%*%beta
  sum(r^2)/2
}
grad = function(beta){
  -crossprod(X,Y-X%*%beta)
}
```

```
beta0=apply(X,2,mean)
opt=optim(par=beta0,fn=loss,gr=grad,method="BFGS")
opt

## $par
## [1]  0.51501717  1.15268959 -0.08365974  0.04815986
##
## $value
## [1] 57.20076
##
## $counts
## function gradient
##       30        7
##
## $convergence
## [1] 0
##
## $message
## NULL

# Standard errors
beta_hat=opt$par
sqrt(diag(sum((Y-X%*%beta_hat)^2)/
            (100-4)*solve(crossprod(X,X))))

## [1] 0.11117243 0.18878526 0.05701767 0.05086372
```

There are less iterations for EM than for BGFS, 7 against approximately 30 here.

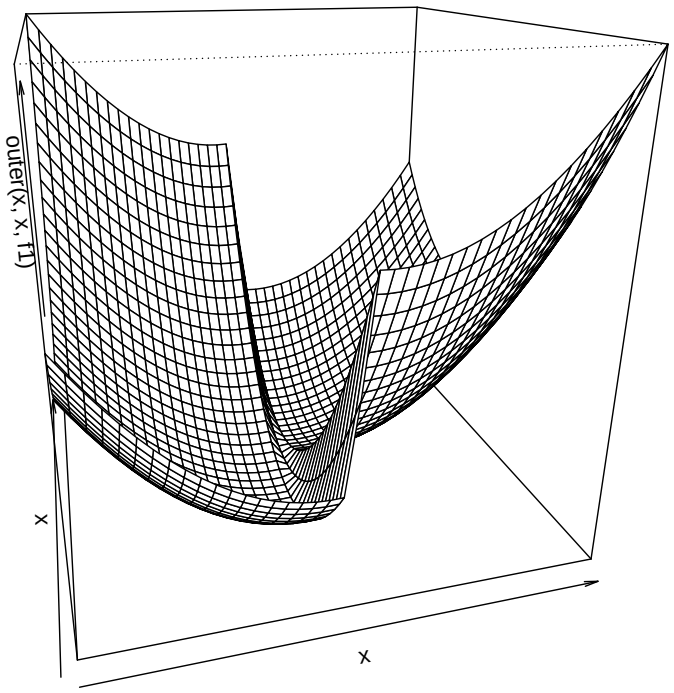
4 Helical valley

I define three slices, one in each direction and plot the corresponding 3d output:

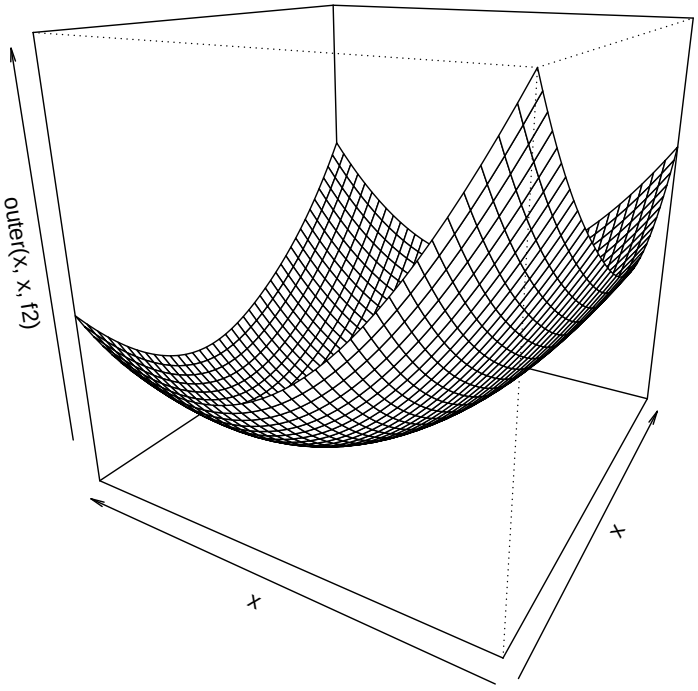
```
# Helical valley
theta <- function(x1,x2) atan2(x2, x1)/(2*pi)

f <- function(x) {
```

```
  f1 <- 10*(x[3] - 10*theta(x[1],x[2]))
  f2 <- 10*(sqrt(x[1]^2+x[2]^2)-1)
  f3 <- x[3]
  return(f1^2+f2^2+f3^2)
}

a=0 #Define slice
f1 = function(x2,x3){
  m=cbind(x2,x3)
  apply(m,1,function(row) f(c(a,row)))
}

b=5 #Define slice
f2 = function(x1,x3){
  m=cbind(x1,x3)
  apply(m,1,function(row) f(c(row[1],b,row[2])))
}

c=5 #Define slice
f3 = function(x1,x2){
  m=cbind(x1,x2)
  apply(m,1,function(row) f(c(row,c)))
}
```

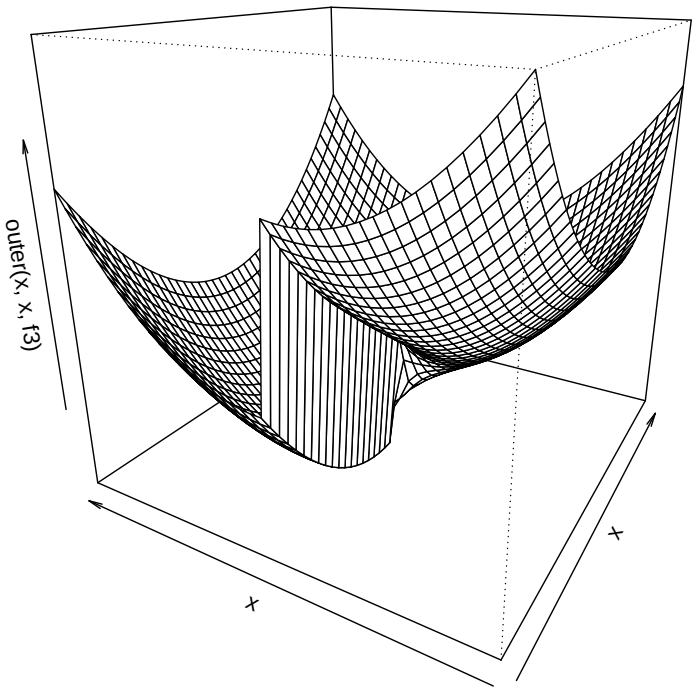
```
x=seq(-10,10,by=0.5)
persp(x,x,outer(x,x,f1),phi=20,theta=-15)
```




```
persp(x,x,outer(x,x,f2),phi=20,theta=-60)
```

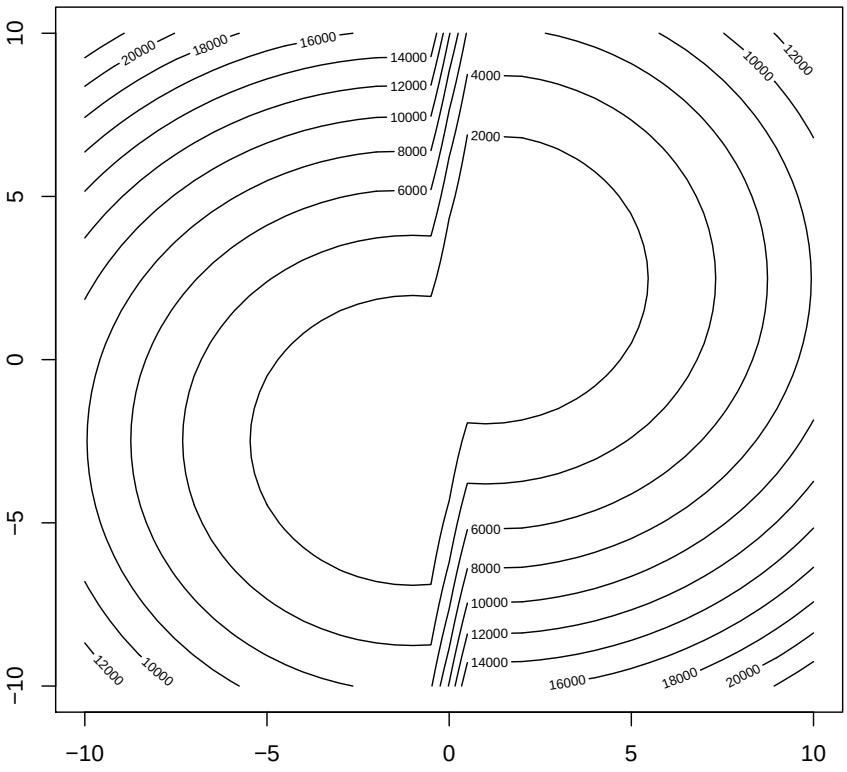



```
persp(x,x,outer(x,x,f3),phi=20,theta=-60)
```



I can also plot a contourplot:

```
contour(x,x,outer(x,x,f1))
```



Finally, here are the results for two different initializations which lead to different outcomes. The first call may be more appropriate since the two methods (optim and lnm) lead approximately to the same result. However, for the second example it seems that another loal minima is reached.

```
# Results
x_init=c(1,1,1)
optim(x_init,f)

## $par
## [1]  0.9999779414 -0.0001349269 -0.0001927127
##
## $value
```

```
## [1] 1.343098e-07
##
## $counts
## function gradient
##      172       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
max(abs(optim(x_init,f)$par-nlm(f,x_init)$estimate)) #similar
```

```
## [1] 6.258705e-05
```

```
# Different starting point
x_init=c(-20,100,1)
optim(x_init,f)
```

```
## $par
## [1]  0.9618505 -0.2799601 -0.4494896
##
## $value
## [1] 0.2025213
##
## $counts
## function gradient
##      101       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
optim(x_init,f)$par-nlm(f,x_init)$estimate #different estimate
```

```
## [1] -0.03814949 -0.27996007 -0.44948964
```