ps8

Malvika Rajeev

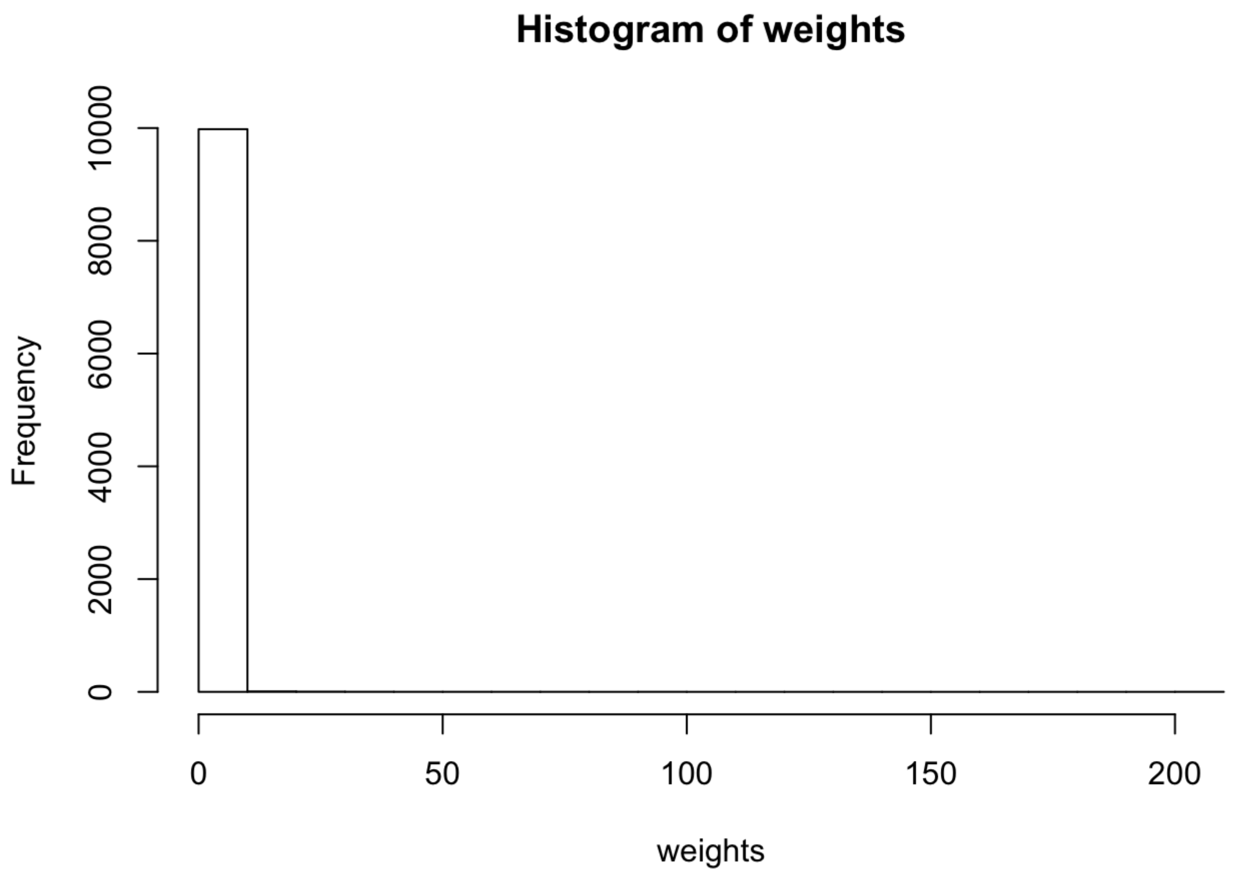
11/26/2018

Question 1.

Sample X from normal distribution centered at -4. target: truncated t-dist, $X < -4$.

PART(A)

```
m <- 10000 #sample size
c <- -4
df <- 3 #degrees of freedom
V <- 1 #variance
#generate M uniform values between 0 and 0.5, and get the inverse cdf.
x <- qnorm(runif(m, 0 , pnorm(c, mean=c)), mean=c)

#get the value of the densities at f and g
f <- dt(x, df=df)
g <- dnorm(x, mean=-4, sd=V)

#standardize everything by CDF at -4
f_n <- f/ pt(q=c, df=3)
g_n <- g/ pnorm(q=c, mean=-4, sd=1)
weights <- f_n/g_n
hist(weights)
```



```
expectedmean <- mean(weights*x)
expectedmean
```

```
## [1] -4.484284
```

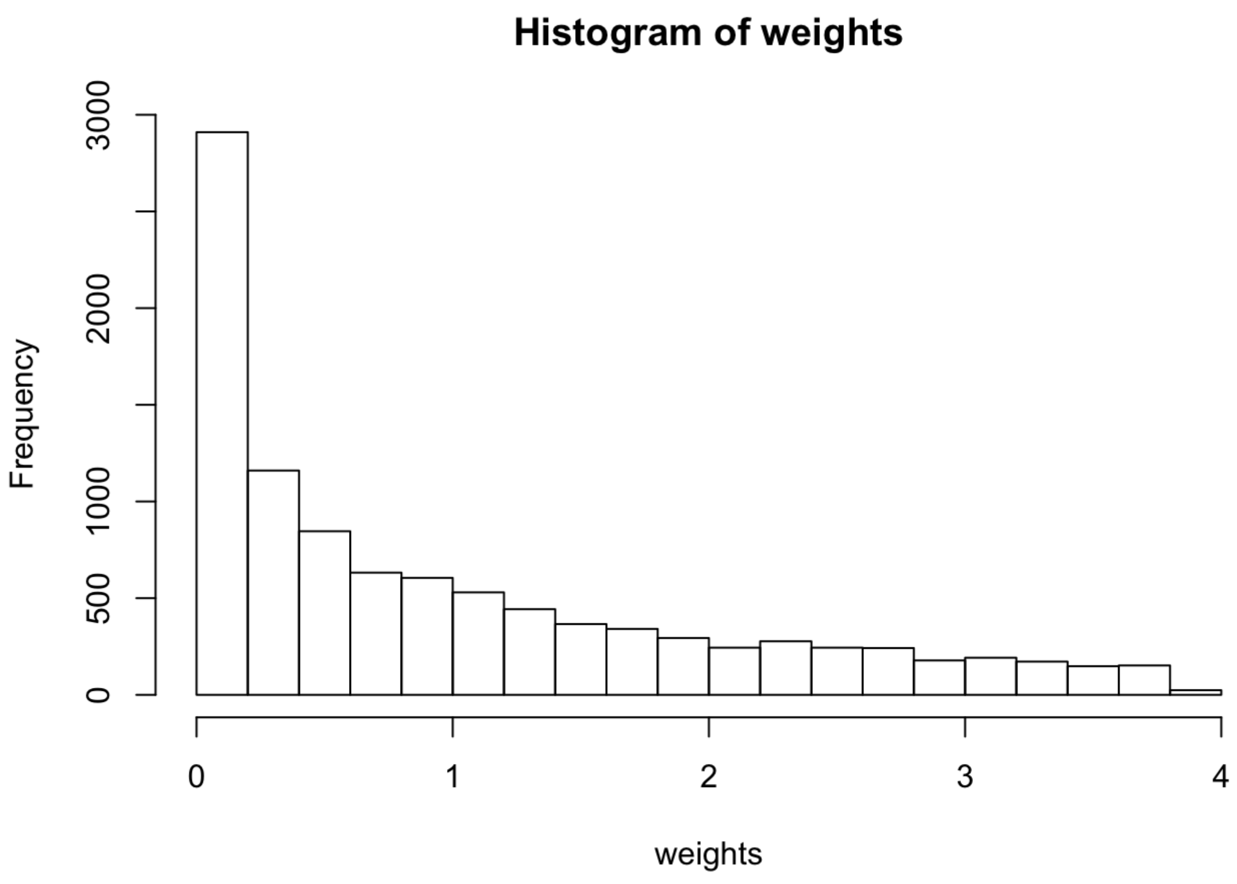
```
var1 <- (1/m)*var(weights*x)
var1
```

```
## [1] 0.04429617
```

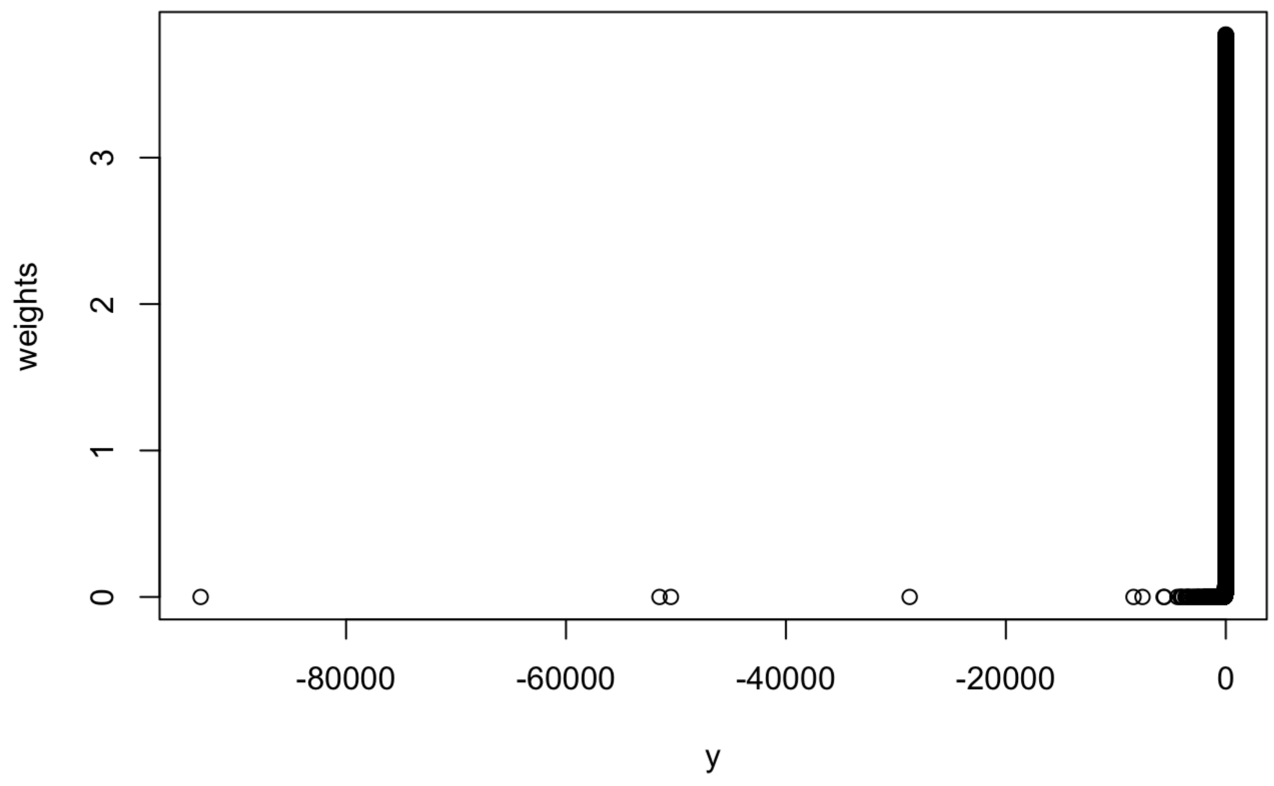
Ideally, we would want that the weights are very large only when X (from the proposal density) is very small. The variance of the first estimator is larger because of some extreme sample points that are pulling it up. The second estimator has more varied weights than the first one does.

PART(B)

```
set.seed(23)
y <- qt( runif(m, 0, pt(c, ncp=c, df=1)),
                      ncp=c, df=1)
f2 <- dt(y, df=df)
g2 <- dt(y, ncp=-4, df=1)
fn2 <- f2 / pt(q=c, df=df) #normalize weights
gn2 <- g2 / pt(q=c, ncp=c, df=1) # normalize weights
weights <- fn2/gn2
hist(weights)
```



```
plot(y, weights)
```



```
expectedmean <- mean(weights*y)
expectedmean
```

```
## [1] -6.247774
```

```
var2 <- (1/m)*var(weights*y)
var2
```

```
## [1] 0.001731872
```

QUESTION 2

Consider the “helical valley” function (see the ps8.R file in the repository). Plot slices of the function to get a sense for how it behaves (i.e., for a constant value of one of the inputs, plot as a 2-d function of the other two). Syntax for image(), contour() or persp() (or the ggplot2 equivalents) from the R bootcamp materials will be helpful. Now try out optim() and nlm() for finding the minimum of this function (or use optimx()). Explore the possibility of multiple local minima by using different starting points.

```
set.seed(2)
library(plotly)
```

```
## Loading required package: ggplot2
```

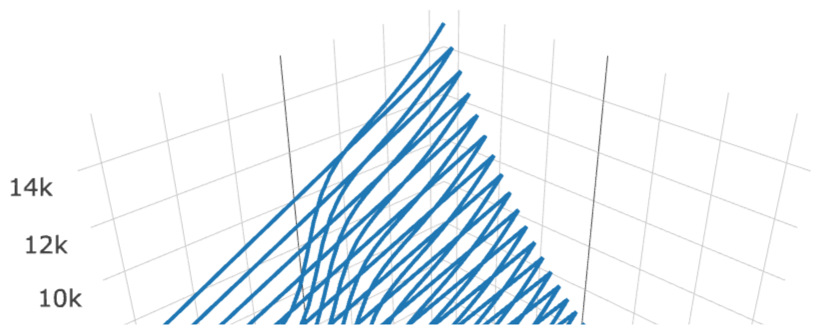
```
##
## Attaching package: 'plotly'
```

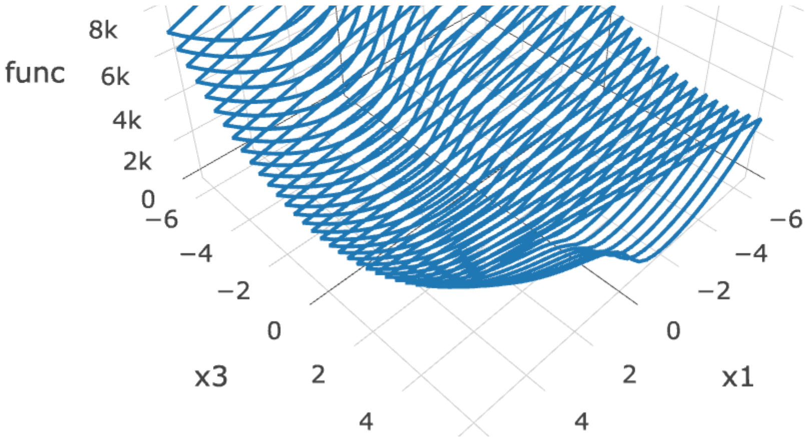
```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

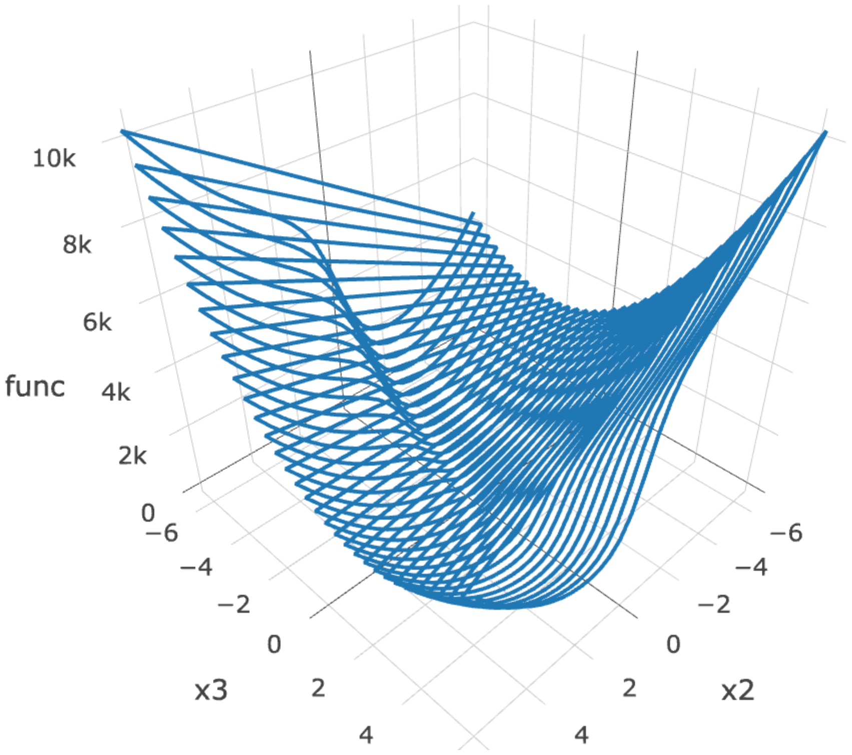
```
## The following object is masked from 'package:graphics':
##
##     layout
```

```r
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)
}

#visualising

s <- seq(-2*pi, 2*pi, by=1/8*pi) # random values


combx3 <- expand.grid(x1=s, x2=s) # get every combination
combx3$x3 <- pi/4 #fixing x3
combx3 <- within(combx3, {
  f1 <- 10*(x3 - 10*theta(x1,x2))
  f2 <- 10*(sqrt(x1^2 + x2^2) - 1)
  f3 <- x3
  func <- f1^2 + f2^2 + f3^2
})

combx1 <- expand.grid(x2=s, x3=s)
combx1$x1 <- pi/4
combx1 <- within(combx1, {
  f1 <- 10*(x3 - 10*theta(x1,x2))
  f2 <- 10*(sqrt(x1^2 + x2^2) - 1)
  f3 <- x3
  func <- f1^2 + f2^2 + f3^2
})

combx2 <- expand.grid(x1=s, x3=s)
combx2$x2 <- pi/4
combx2 <- within(combx2, {
  f1 <- 10*(x3 - 10*theta(x1,x2))
  f2 <- 10*(sqrt(x1^2 + x2^2) - 1)
  f3 <- x3
  func <- f1^2 + f2^2 + f3^2
})

plot_ly(combx2, x = ~x1, y = ~x3, z = ~func, type = 'scatter3d', mode = 'lines', line
 = list(width = 4))
```

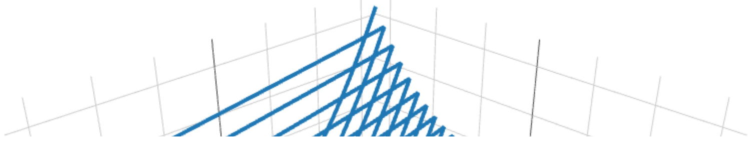


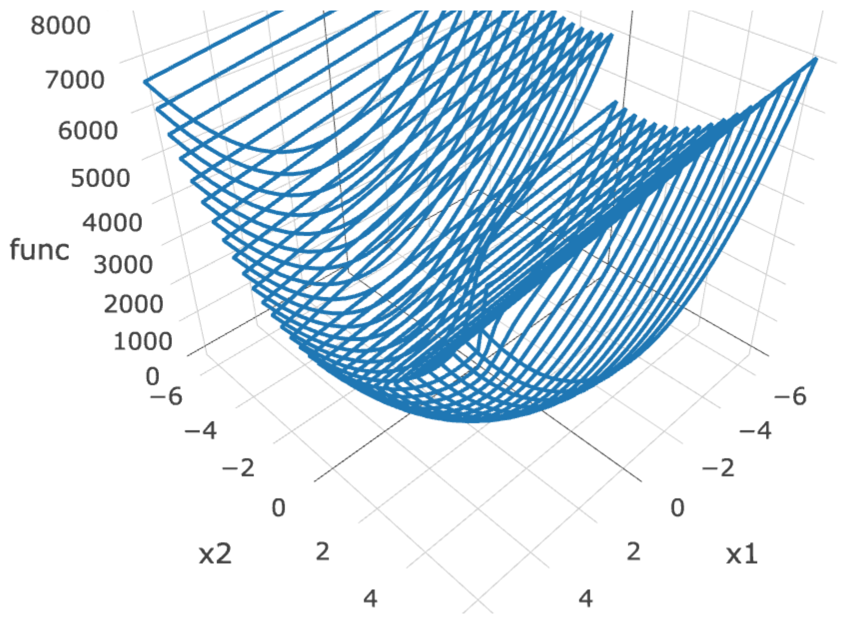


```
plot_ly(combx1, x = ~x2, y = ~x3, z = ~func, type = 'scatter3d', mode = 'lines', line
 = list(width = 4))
```



```
plot_ly(combx3, x = ~x1, y = ~x2, z = ~func, type = 'scatter3d', mode = 'lines', line
 = list(width = 4))
```





```
##optimization using optim and nlp
opt_methods <- c("BFGS", "CG", "L-BFGS-B", "Nelder-Mead", "SANN")
minval <- rep(NA, length(opt_methods))
for (i in 1:length(opt_methods)){
  cat(opt_methods[i],'results')
  min <- optim(par=c(0.1,0.11,0.12), fn=f, method=opt_methods[i])
  cat('\nThe minimizers are ', min$par)
  cat('\nThe min is ', min$value,'\n')
  minval[i] <- min$value
}
```

```
## BFGS results
## The minimizers are  1 -8.934041e-11 -1.425899e-10
## The min is  2.071732e-20
## CG results
## The minimizers are  0.999978 0.00232571 0.003678713
## The min is  1.346759e-05
## L-BFGS-B results
## The minimizers are  1 -5.581213e-07 7.46522e-08
## The min is  9.385675e-11
## Nelder-Mead results
## The minimizers are  1.000008 -2.135307e-05 1.912592e-05
## The min is  2.888422e-07
## SANN results
## The minimizers are  1.000446 0.005433781 0.01430217
## The min is  0.003427095
```

```
###Now NLM

nlpMin <- nlm(f=f, p=c(0.1,0.11,0.12))
print(nlpMin)
```

```
## $minimum
## [1] 7.510908e-21
##
## $estimate
## [1] 1.000000e+00 4.523708e-11 7.293248e-11
##
## $gradient
## [1]  9.174439e-10 -2.977560e-10  3.329504e-10
##
## $code
## [1] 1
##
## $iterations
## [1] 24
```

```
cat("Does NLP have a lower minimum?", (min(minval) < nlpMin$minimum))
```

```
## Does NLP have a lower minimum? FALSE
```

QUESTION 3

To fit the model $Y_i = Ber(\Phi(X_i^T\beta))$. We can write the complete data model as: $Y_i|Z_i, \beta = I(Z_i > 0)$ and $Z_i|\beta = N(x_i^T\beta, 1)$.

Also $P(Yi = 1|\beta) = \int P(Yi = 1, Zi|\beta)dZi$

$P(Z_i > 0|\beta) = P(Z_i - x_i^T\beta > -x_i^T\beta)$

$1 - \Phi(-x_i^T\beta) = \Phi(x_i^T\beta)$

And after computing the Q-function we can see that

$Z_i|y_i = 0, \beta^{(t)} = TN(x_i^T\beta^t, 1; (\infty, 0))$

$Z_i|y_i = 1, \beta^{(t)} = TN(x_i^T\beta^t, 1; (0, \infty)$

So E step is: $Z_i^{t+1} = x_i\beta^{(t)} - \frac{\phi(x_i\beta^{(t)})}{\Phi(-x_i\beta^{(t)}))}$, if y_i = 0 And

$Z_i^{t+1} = x_i\beta^{(t)} + \frac{\phi(x_i\beta^{(t)})}{1-\Phi(-x_i\beta^{(t)}))}$, if y_i = 1.

And the M step is to maximimise:

$\beta^{(t+1)} = (X^TX)^{-1}X^TZ^{(t+1)}$ which can be done efficiently using QR decomposition.

Also, the function can use input argument as tolerance level.

---

a.

```r
set.seed(2)

##the actual function
##the actual function
em_function <- function(betastart, X, Y, maxits, tol = 0.00001){
  iterations <- 0
  beta_current = betastart
  M <- X %*% beta_current #Mean
  converged <- FALSE
  Z <- rnorm(length(Y), mean=M)

  # QR Decomposition for OLS
  # using QR decomposition to get the Q, R matrix

  X.qr <- qr(X)
  Q <- qr.Q(X.qr)
  R <- qr.R(X.qr)

  while((!converged) && iterations < maxits){
    # run this until iterations done or convergence occurs
    ##E-STEP
    Z_t_p = Z
    Z = ifelse(Y==1, M + dnorm(M)/pnorm(M),
               M - dnorm(M)/pnorm(-M)) # Define Z as per value of Y (1 or 0)

    varZ <- ifelse(Y==1, 1 - M*dnorm(M)/pnorm(M)-(dnorm(M)/pnorm(M))^2,
                      1 + M*dnorm(-M)/pnorm(-M)-(dnorm(-M)/pnorm(-M))^2)
    ## formula for variance of truncated normal

    # M step using QR decomposition that we've done

    beta_new = backsolve(R,crossprod(Q,Z))
    M = X %*% beta_new
    ll = -(1/2)*sum(varZ + Z^2) - # M^2 + sigma^2 ##log-likelihood
                  (1/2)*crossprod(M) + #t(M) * M
                    crossprod(Z , M) - # t(Z) * M
                      n*log(sqrt(2*pi))

    converged <- (max(abs(Z_t_p - Z)) <= tol)
    iterations <- iterations + 1
  }
  if (converged == TRUE){print("Convergence reached")} else {print("Convergence not r
eached")}
  return(list(ll=ll, startingval = betastart, betas=beta_new, converged=converged,

              iterations=iterations))

}
```

b. For starting values I took $\beta = (0.25, 0.25, 0, 0)$. It’s convenient to take $\beta_i = \frac{1}{c}$, where $\Sigma = c$.

```
##(c)


set.seed(2)

n <- 100
#generating data
sample_data <- data.frame(x1 = rnorm(n), x2 = rnorm(n),
                          x3= rnorm(n), y = sample(c(0,1),n
                                                    ,replace=TRUE))

##adding noise
sample_data$x1 <- sample_data$x1 + sample_data$y


probit <- glm(y~., data=sample_data, family=binomial(link = "probit"))
summary(probit)
```

```
##
## Call:
## glm(formula = y ~ ., family = binomial(link = "probit"), data = sample_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0741  -0.9037   0.4071   1.0612   1.6232
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.2628     0.1486  -1.768    0.077 .
## x1            0.4946     0.1190   4.156 3.24e-05 ***
## x2            0.1894     0.1384   1.369    0.171
## x3            0.1985     0.1285   1.545    0.122
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 138.59  on 99  degrees of freedom
## Residual deviance: 114.74  on 96  degrees of freedom
## AIC: 122.74
##
## Number of Fisher Scoring iterations: 5
```

```
X <- as.matrix(cbind(1, sample_data[,1:3])) ##add a column of 1s for beta zero
Y <- sample_data$y

beta_start <- c(0.25,0.25,0,0)
test <- em_function(betastart = beta_start, X=X, Y=Y, maxits =500)
```

```
## [1] "Convergence reached"
```

```
test$betas
```

```
##             [,1]
## [1,] -0.2628258
## [2,]  0.4945872
## [3,]  0.1894303
## [4,]  0.1984833
```

```
print(round(test$betas - coef(probit)))
```

```
##      [,1]
## [1,]    0
## [2,]    0
## [3,]    0
## [4,]    0
```

d.

```
set.seed(1)
loglikelihood <- function(beta, X, Y){
  M = X %*% beta
  ll = sum(Y*pnorm(M, log.p=T) + (1-Y)*pnorm(-M, log.p=T))
  -ll
} ##taking the negative of loglikelihood for optim



optim_values <- optim(par=c(0.25,0.25,0,0), fn=loglikelihood,     X=X,Y=Y,method="BFG
S",
                      control=list(trace=TRUE))
```

```
## initial  value 65.292702
## iter  10 value 57.367704
## iter  10 value 57.367704
## final  value 57.367704
## converged
```

```
print(optim_values)
```

```
## $par
## [1] -0.2628268  0.4945893  0.1894318  0.1984829
##
## $value
## [1] 57.3677
##
## $counts
## function gradient
##       23       10
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
print(optim_values$par)
```

```
## [1] -0.2628268  0.4945893  0.1894318  0.1984829
```

```
print(optim_values$counts)
```

```
## function gradient
##       23       10
```

```
cat("Minimizing the loglikelihood reached convergence in ",optim_values$counts[1],"it
erations. EM algorithm reached convergence in", test$iterations,"iterations.")
```

```
## Minimizing the loglikelihood reached convergence in  23 iterations. EM algorithm r
eached convergence in 22 iterations.
```