ps8

Yihuan Song

11/19/2018

question1

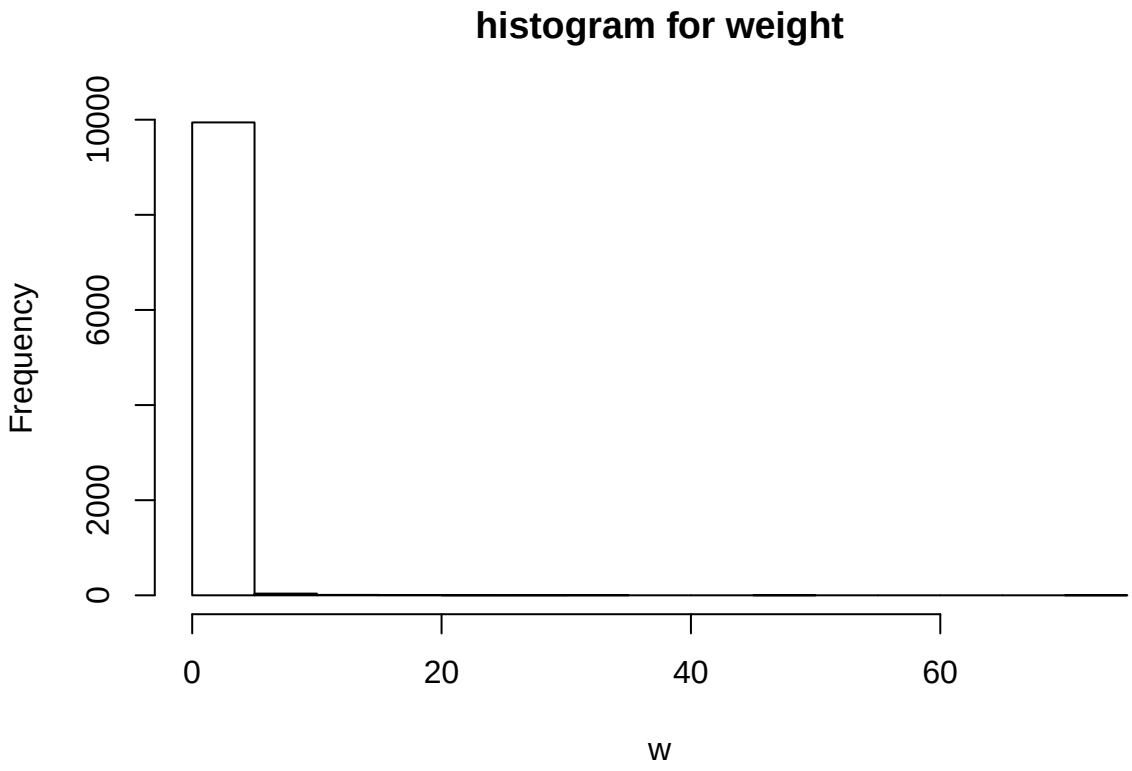
(a)

Using a sampling density of a normal distribution centered at -4 and truncated, we can see from the histogram that the var($\hat{\theta}$) is large, since the histogram is very skewed and shows that the weights exist very extreme values. The var($\hat{\theta}$) is 0.008996.

```
m <- 10000 # number of samples for each estimator
set.seed(0)
x <- -abs(rnorm(m)) - 4 # sample from g(x) being a half-normal distribution centered at -4
f <- dt(x, df = 3) / pt(-4, 3)  # density of x under f
g <- 2*dnorm(x, mean = -4, sd =1, log = FALSE)   # density of x under g
w <- f / g  # weights
max(w) # detect outlier
```

```
## [1] 72.40674
```

```
#create histogram of weight
hist(w, main = "histogram for weight")
```




```
mean <- mean(x*w)
mean
```

```
## [1] -4.246086
```

```
var <- var(x*w) / m  # variance of IS estimator
var
```

```
## [1] 0.008996065
```

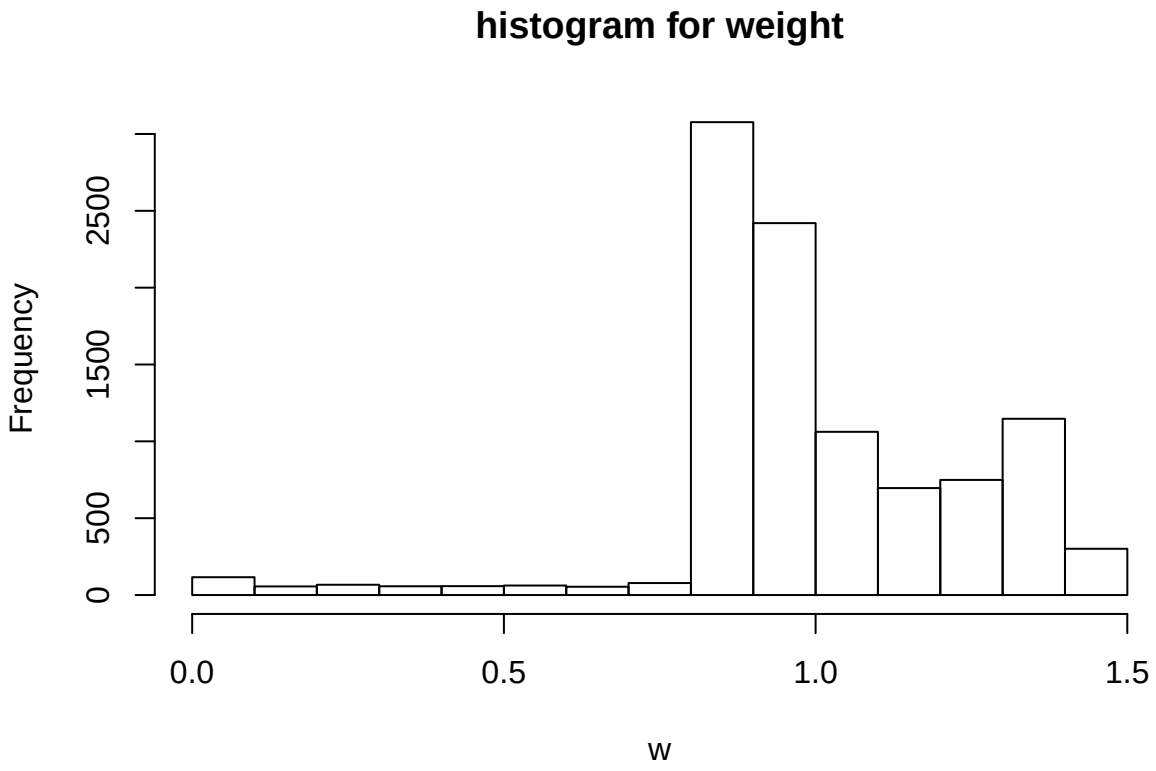
(b)

Using a sampling density of a t distribution, with 1 degree of freedom, centered at -4 and truncated, we can see from the histogram that the var($\hat{\theta}$) is small, since the histogram shows that the weights are not extreme. The var($\hat{\theta}$) is 0.000924, which is better than the estimation in part (a).

```
m <- 10000 # number of samples for each estimator
set.seed(0)
x <- -abs(rt(m, df = 1)) -4 # sample from g(x) being a half-normal distribution centered at -4
f <- dt(x, df = 3) / pt(-4, 3)  # density of x under f
g <- 2*dt(x + 4, df = 1)   # density of x under g
w <- f/g  # weights
max(w)
```

```
## [1] 1.405829
```

```
#create histogram of weight
hist(w, main = "histogram for weight")
```




```
mean <- mean(x*w)
mean
```

```
## [1] -6.208823
```

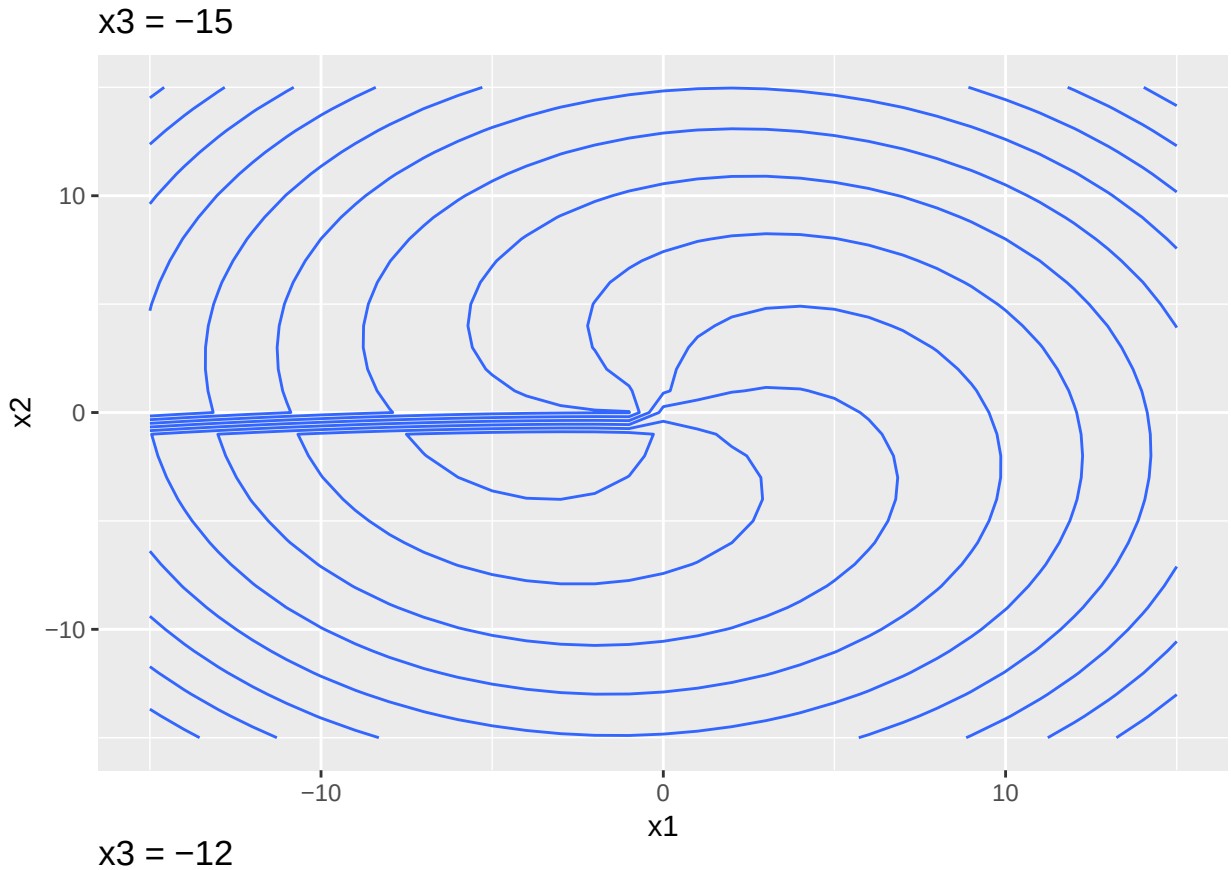
```
var <- var(x*w) / m    # variance of IS estimator
var
```

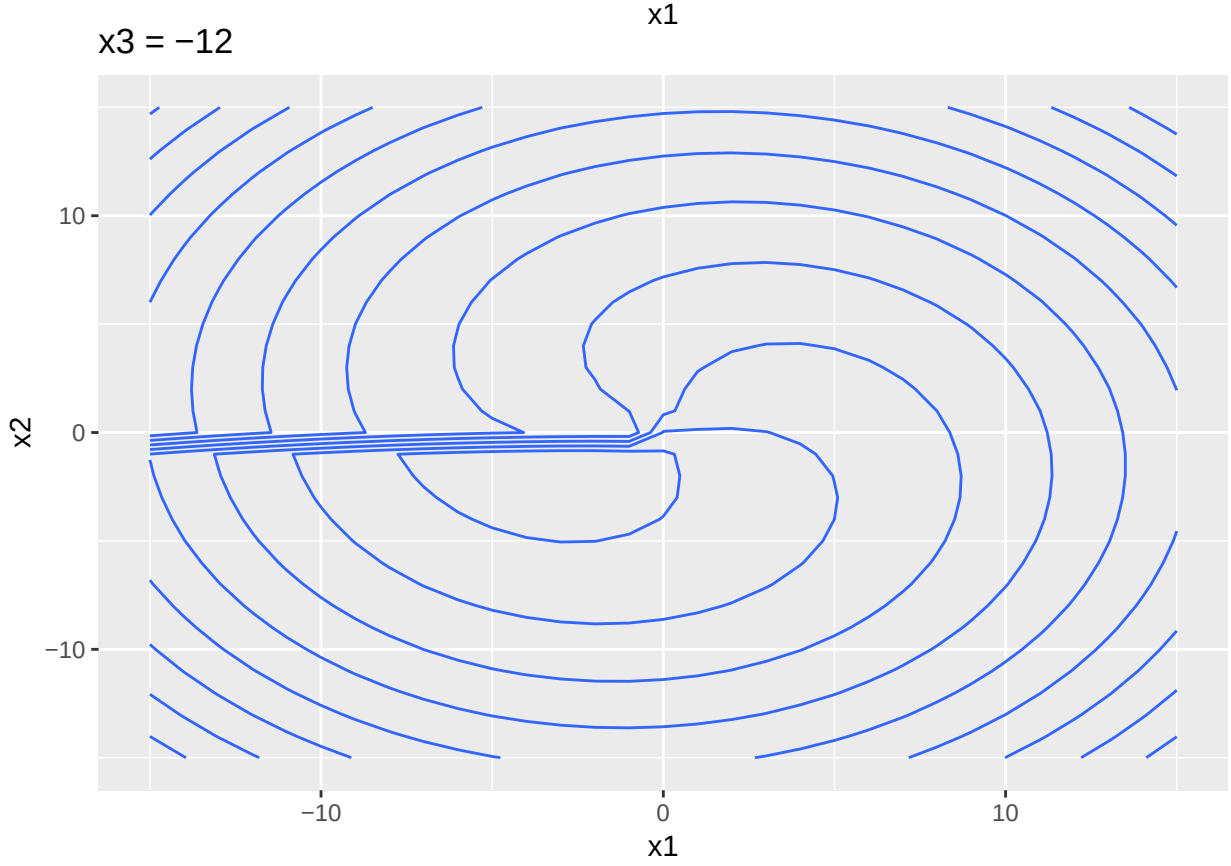
```
## [1] 0.0009242501
```

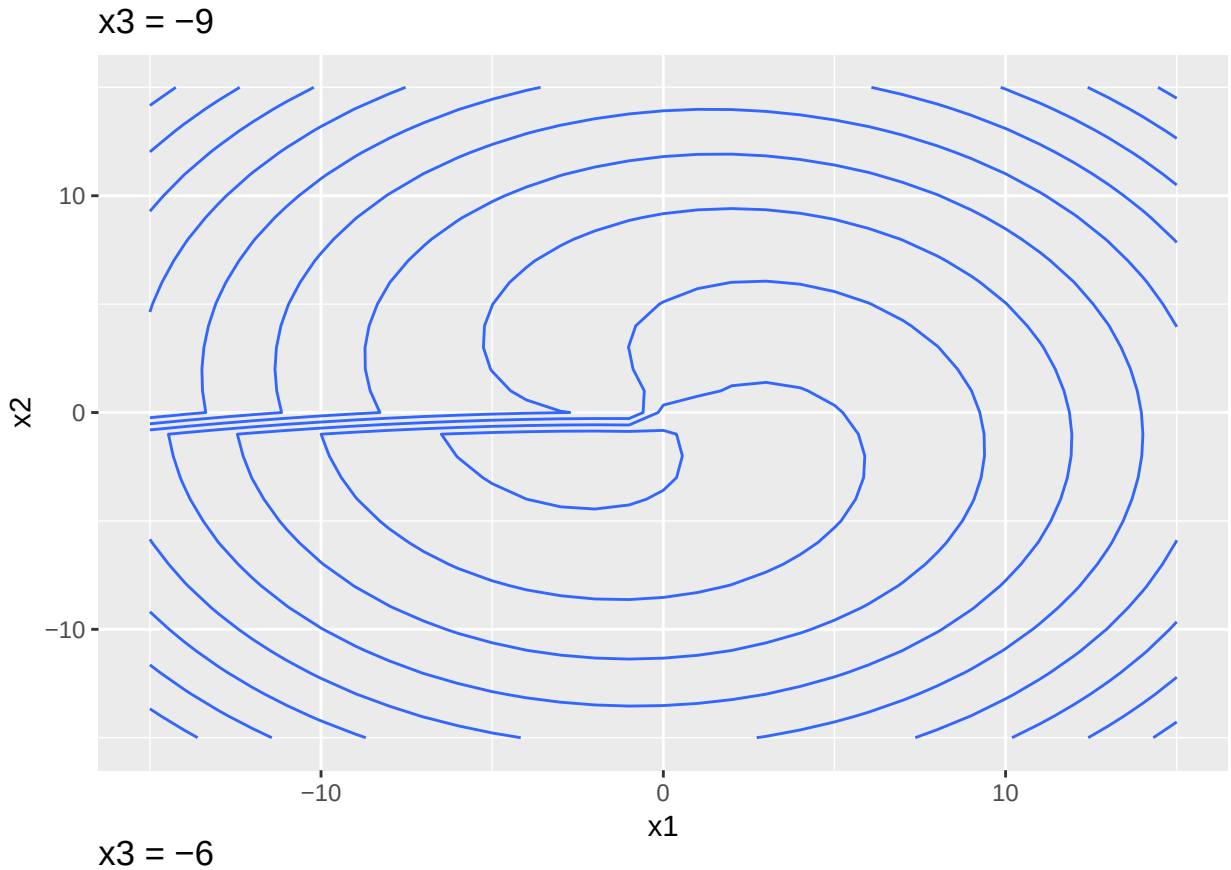
question2

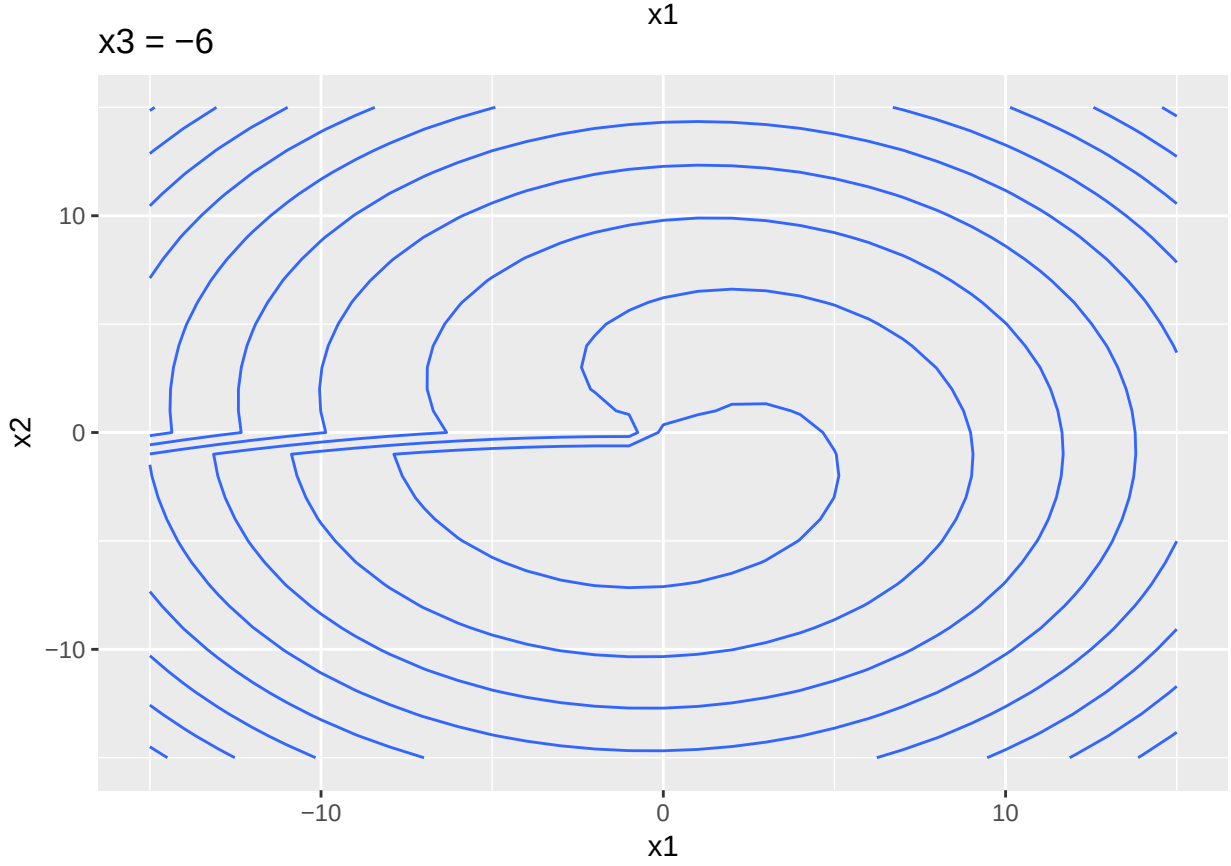
Using ggplot's contour function, I ploted how the function of variables x1 and x2 behaves when x3 is set as a sequence of constants. From the plot, we can see that the minimum might occur at around 0. Then, using the "BFGS" method and "Nelder-Mead" method for optim() and using nlm(), the result suggested that the minimum goes to 0, as the (x1,x2,x3) goes to values (1,0,0). Testing out for different starting values suggested that most results converged to the same minimum(which is 0), few result turned out positive values for the "Nelder-Mead" method, and overall it suggested that 0 is the global minimum for f() as (x1,x2,x3) goes to (1,0,0).

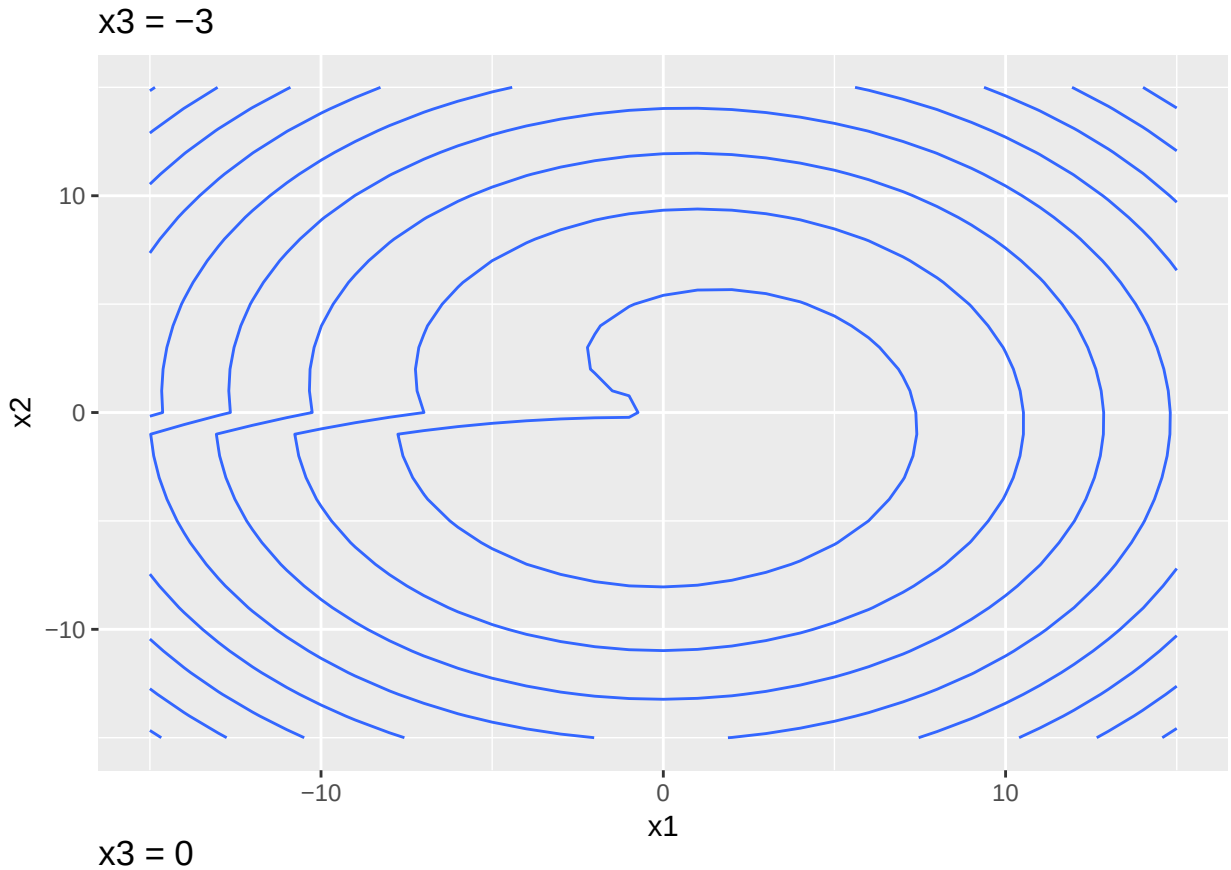
```
library(ggplot2)
theta <- function(x1,x2) atan2(x2, x1)/(2*pi)
f <- function(x) {
  f1 <- 10*(x[3] - 10*theta(x[1],x[2]))
  f2 <- 10*(sqrt(x[1]^2 + x[2]^2) - 1)
  f3 <- x[3]
  return(f1^2 + f2^2 + f3^2)
}
#generate x1 and x2 variables
x1 <- seq(-15, 15)
x2 <- seq(-15, 15)
par(mfrow = c(3,4))
for(x3 in seq(-15, 15, by = 3)){
  #combinations of x1 and x2
  comb <- expand.grid(x1, x2)
  #generate (x1,x2,x3) to pass to f()
  df <- cbind(comb,rep(x3, nrow(comb)))
  vals <- apply(df, 1, f)
  #generate dataframe containing x1,x2 and the function values
  #for the contour function
  all <- cbind.data.frame(comb, vals)
  names(all) <- c("x1", "x2", "value")
  print(ggplot(data = all, aes(x = x1, y = x2, z = value)) +
            geom_contour() + ggtitle(paste0('x3 = ',x3)))
}
```

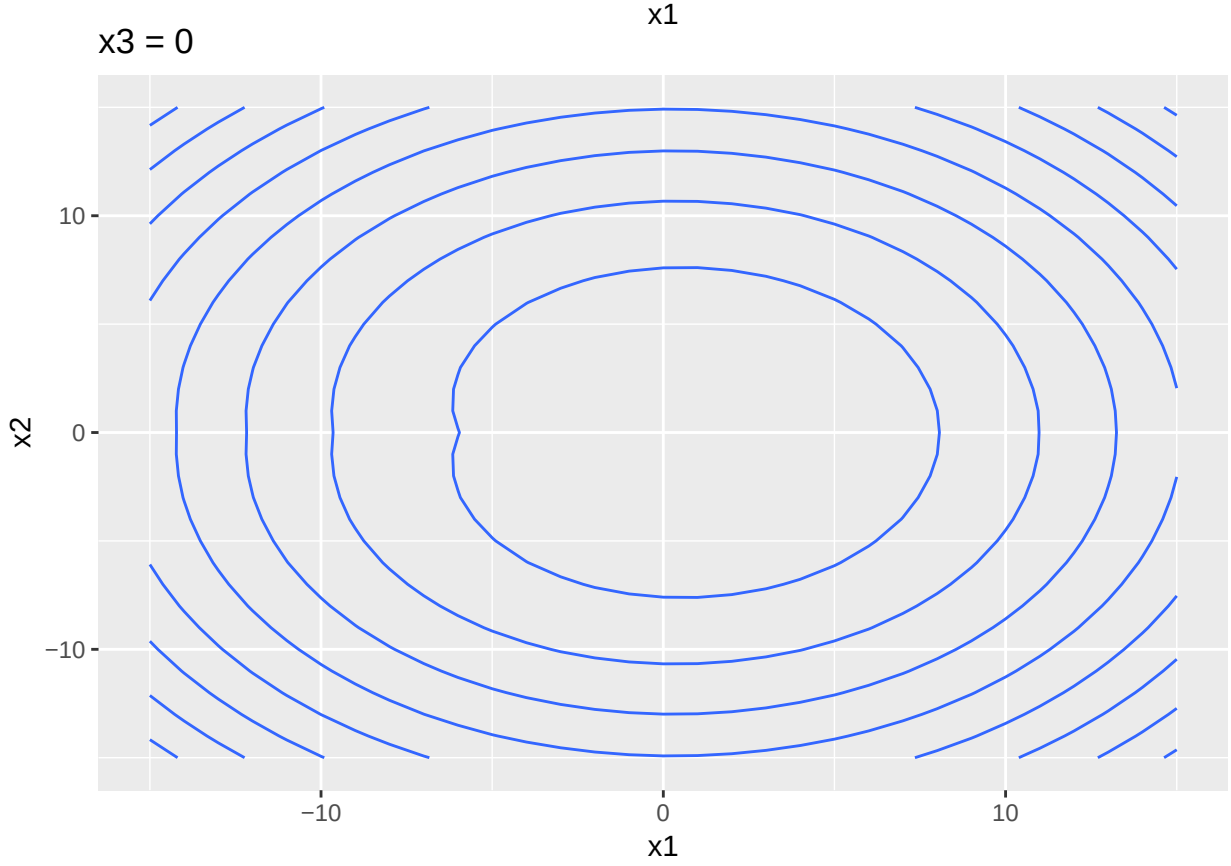

x3 = −15
10
0
−10
x2
−10
0
10
x1

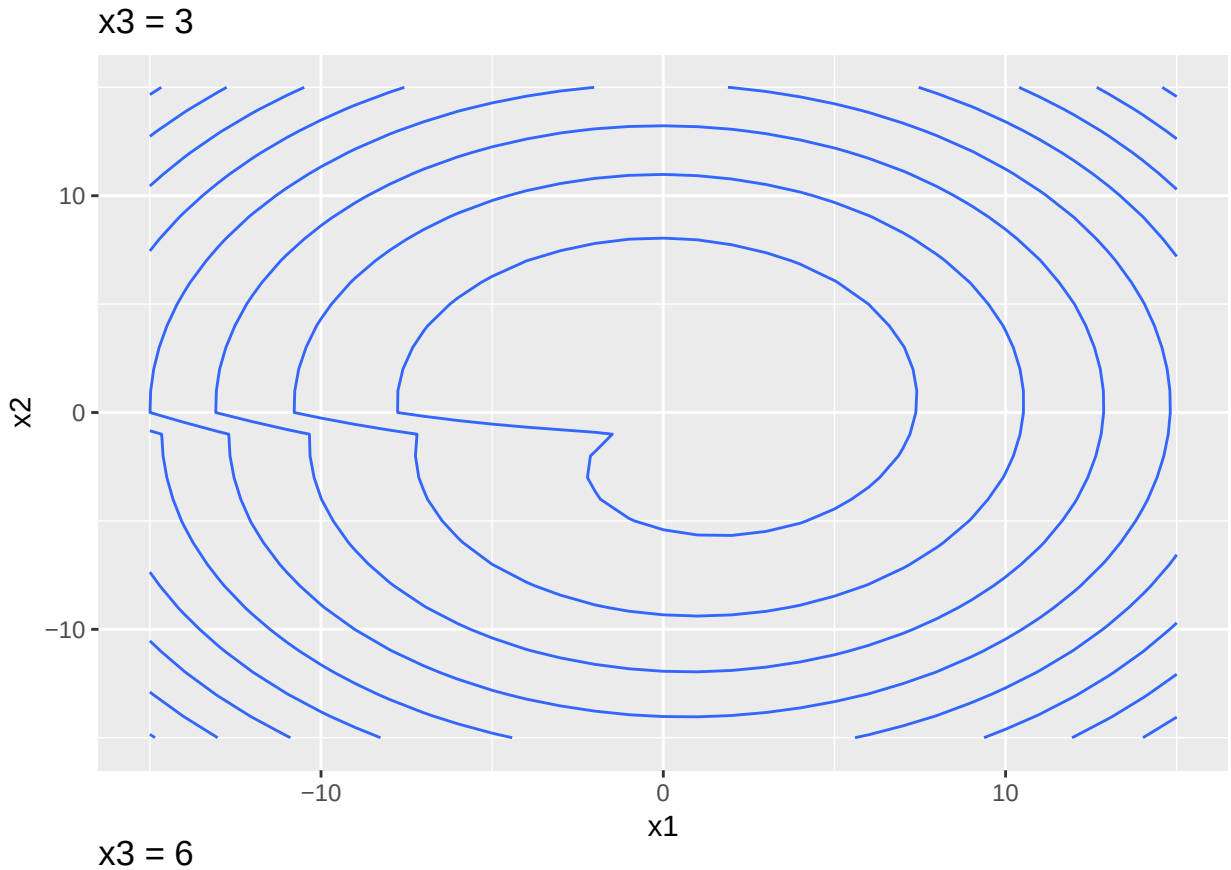

x3 = −12
10
0
−10
x2
−10
0
10
x1

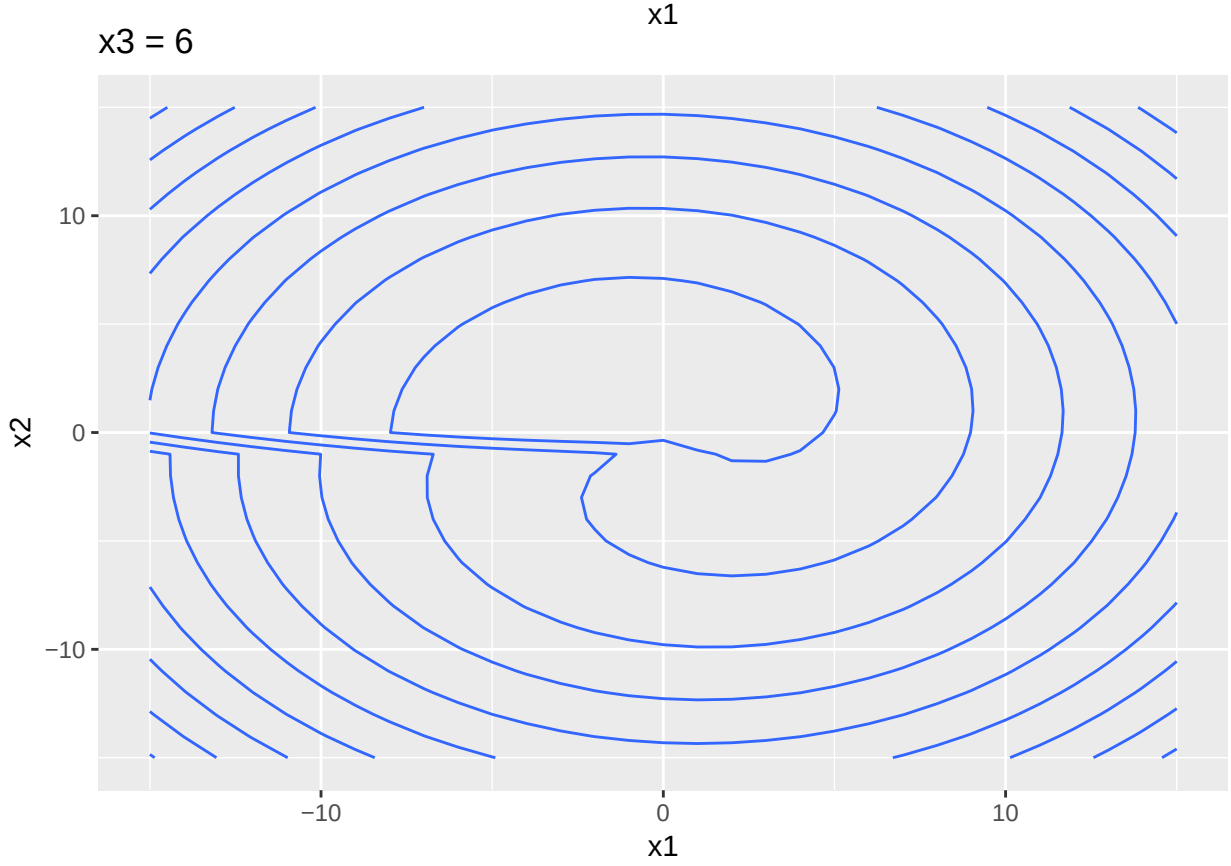

x3 = −9
10
0
−10
x2
−10
0
10
x1

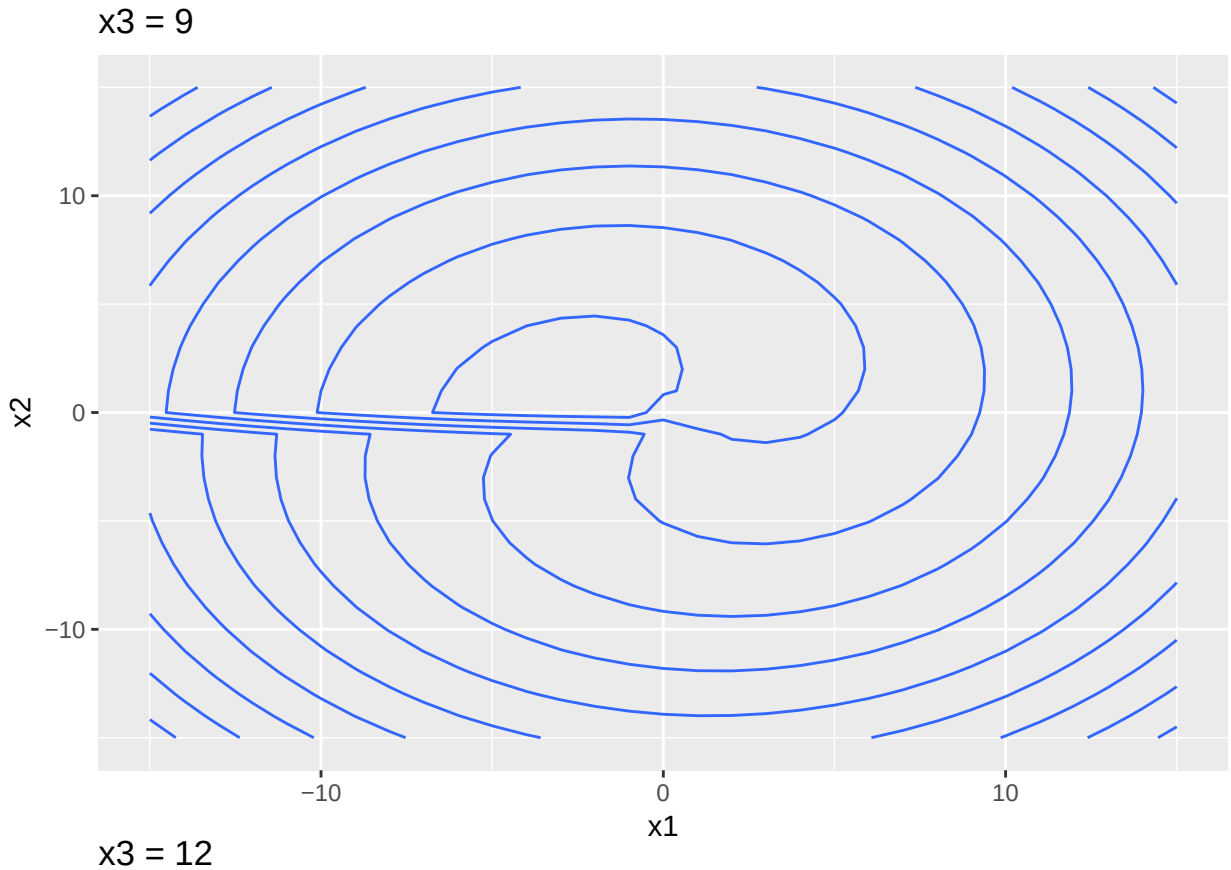

x3 = −6
10
0
−10
x2
−10
0
10
x1

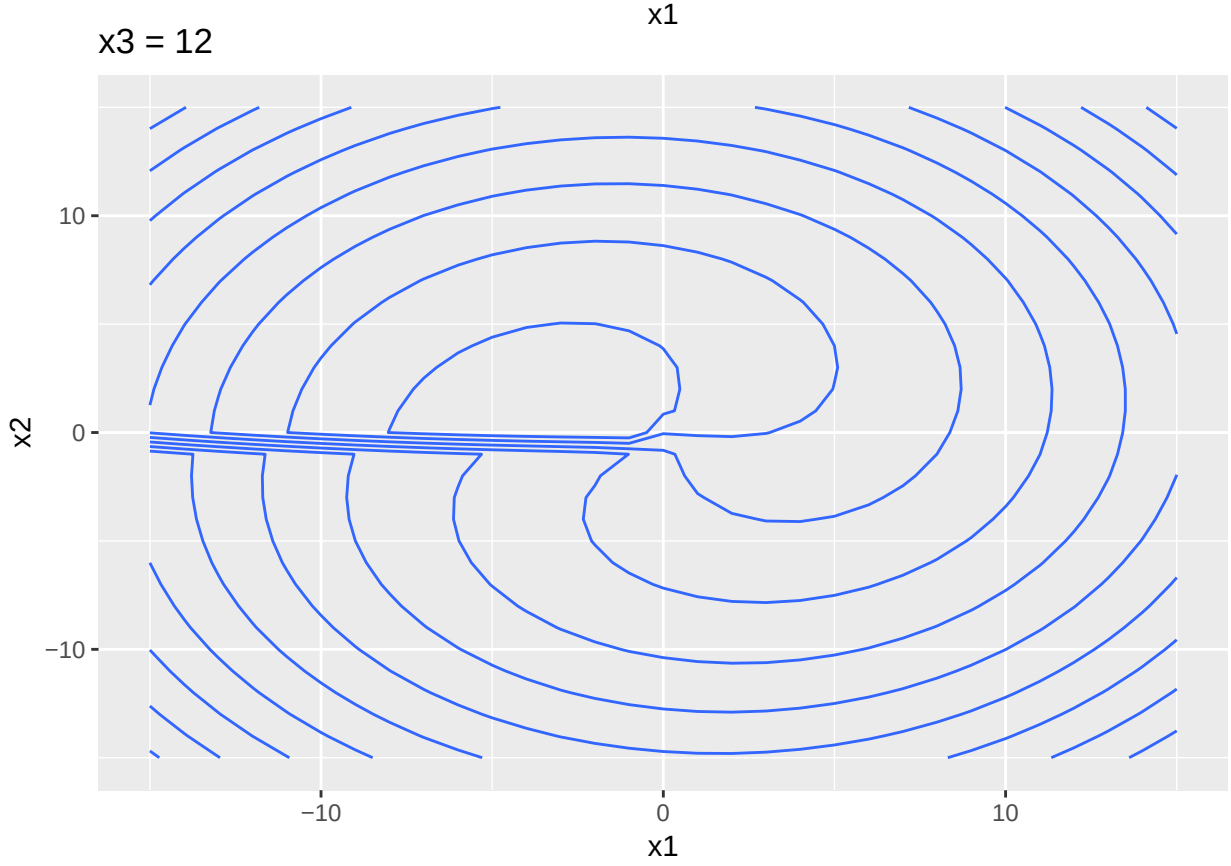

x3 = −3
10
0
−10
x2
−10
0
10
x1

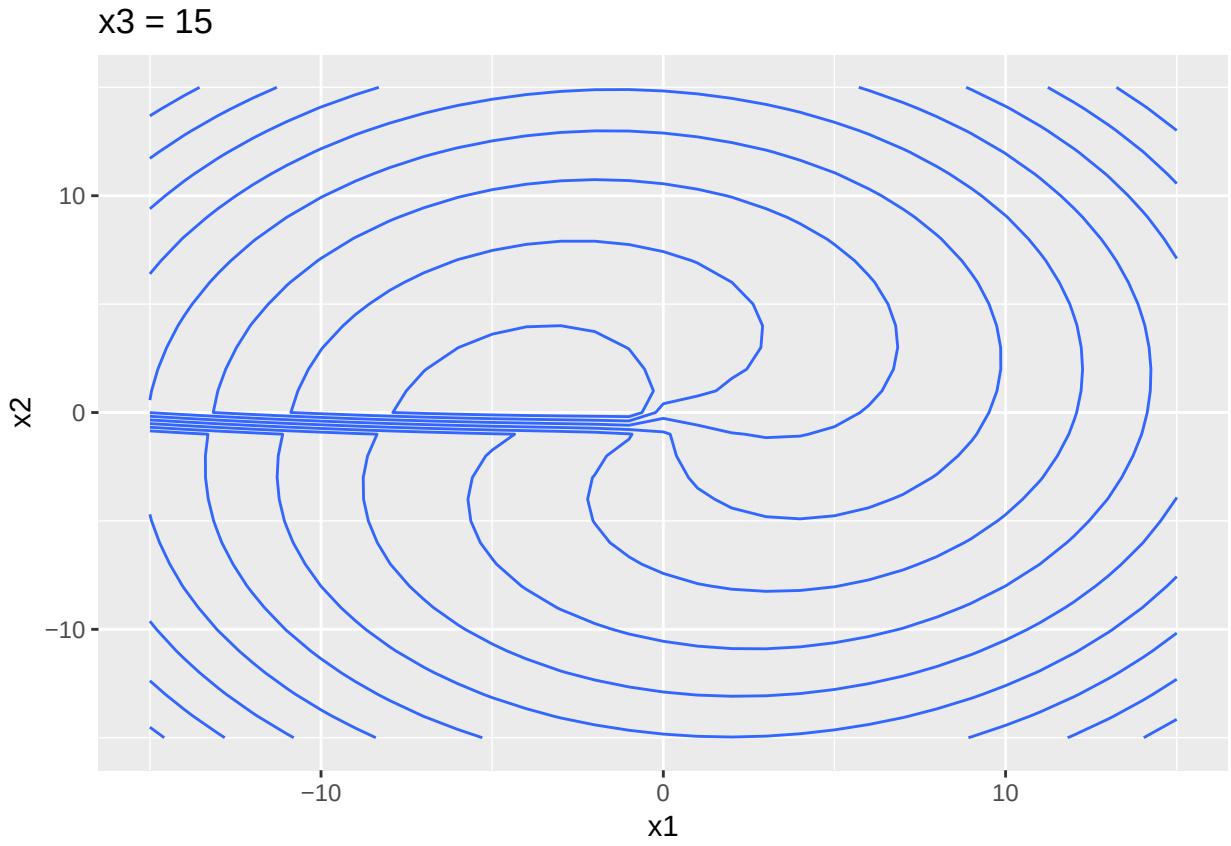

x3 = 0
10
0
−10
x2
−10
0
10
x1


x3 = 3
10
0
−10
x2
−10
0
10
x1


x3 = 6
10
0
−10
x2
−10
0
10
x1


x3 = 9
10
0
−10
x2
−10
0
10
x1


x3 = 12
10
0
−10
x2
−10
0
10
x1




```
##first set of starting points
#use optim() with Nelder-Mead
set.seed(1)
stp <- runif(3,-10,10)
optim(stp, f)
```

```
## $par
## [1]  0.9998451578 -0.0007054304 -0.0006690104
##
## $value
## [1] 2.343915e-05
##
## $counts
## function gradient
##      250       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
#use optim() with BFGS
optim(stp, f, method = 'BFGS')
```

```
## $par
## [1] 1.000000e+00 2.608788e-12 4.141869e-12
##
```

```
## $value
## [1] 1.721664e-23
##
## $counts
## function gradient
##       73       28
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
#use nlm()
nlm(f, p = stp)
```

```
## $minimum
## [1] 3.23757e-17
##
## $estimate
## [1]  1.000000e+00  1.377693e-10 -3.455681e-10
##
## $gradient
## [1]  1.187361e-08  1.797925e-07 -1.136581e-07
##
## $code
## [1] 1
##
## $iterations
## [1] 27
```

```
##second set of starting points
#use optim() with Nelder-Mead
set.seed(2)
stp <- runif(3,-10,10) #different starting points
optim(stp, f)
```

```
## $par
## [1]  1.000187294 -0.001750213 -0.002908657
##
## $value
## [1] 1.355427e-05
##
## $counts
## function gradient
##      244       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
#use optim() with BFGS
optim(stp, f, method = 'BFGS')
```

```
## $par
## [1]  1.000000e+00 -1.078094e-12 -2.343025e-12
##
## $value
## [1] 5.538787e-21
##
## $counts
## function gradient
##       73       26
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
#use nlm()
nlm(f, p = stp)
```

```
## $minimum
## [1] 1.701042e-08
##
## $estimate
## [1]  9.999995e-01 -8.223473e-05 -1.300851e-04
##
## $gradient
## [1] -2.611988e-09  5.033907e-08 -4.158638e-08
##
## $code
## [1] 1
##
## $iterations
## [1] 27
```

```
##third set of starting points
stp <- runif(3,-1000,1000) #different starting points
optim(stp, f)
```

```
## $par
## [1]  0.01589334 -0.91998299 -2.52499832
##
## $value
## [1] 7.289221
##
## $counts
## function gradient
##      176       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
#use optim() with BFGS
optim(stp, f, method = 'BFGS')
```

```
## $par
## [1]  1.000000e+00 -2.144094e-14 -2.782020e-14
##
## $value
## [1] 1.20143e-26
##
## $counts
## function gradient
##      106       42
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
#use nlm()
nlm(f, p = stp)
```

```
## $minimum
## [1] 1.700909e-08
##
## $estimate
## [1]  9.999995e-01 -8.223157e-05 -1.300799e-04
##
## $gradient
## [1] -3.888464e-09 -6.999736e-09  4.844048e-09
##
## $code
## [1] 1
##
## $iterations
## [1] 27
```

question3

(a)

3.(a) observed data: Y ; missing data: Z ; completed data: $[Y, Z]$

$Z \sim N(X_i^T\beta, 1)$;

$y_i = \begin{cases} 1 & \text{if } z_i > 0 \\ 0 & \text{if } z_i \le 0 \end{cases} \sim \text{Ber}(p_i)$, $p_i = \Phi(x_i^T\beta)$

E-step:

$Q(\beta|\beta^t) = E(\log L(\beta|Y,Z)|Y,\beta^t)$

So first, $\log L(\beta|Y,Z)$

$= \log \prod f(y_i, z_i|\beta)$

$= \log(\prod \frac{1}{\sqrt{2\pi}} \exp\{-\frac{1}{2}(z_i - x_i^T\beta)^2\} \cdot \prod [\Phi(x_i^T\beta)]^{y_i} [1 - \Phi(x_i^T\beta)]^{1-y_i})$ ↳ dropped since only z_i's are unknown

$= -\frac{1}{2}\sum (z_i - x_i^T\beta)^2 + \text{constant}$

so $Q(\beta|\beta^t) = E[-\frac{1}{2}\sum (z_i - x_i^T\beta)^2 | y, \beta^t] + \text{constant}$

$= -\frac{1}{2} E[(Z - X\beta)^T(Z - X\beta)|y, \beta^t] + \text{constant}$

$= -\frac{1}{2} E[(Z^TZ - Z^TX\beta - \beta^TX^TZ + \beta^TX^TX\beta)|y, \beta^t] + \text{constant}$

M-step:

$\max_\beta Q(\beta|\beta^t)$:

So the we could take derivative and set it to 0 to get maximum Q.

$\frac{dQ}{d\beta} = 0 + \frac{d}{d\beta}(E(Z^TX\beta|y,\beta^t)) + \frac{d}{d\beta}(\beta^TX^TX\beta)$

$= E(Z|y,\beta^t)^TX - X^TX\beta = 0$

So $\beta^{t+1} = (X^TX)^{-1}X^TE[Z|y,\beta^t]$

Let $E(Z|y,\beta^t) = Z^{t+1}$

then β^{t+1} is the least square estimate when regress Z^{t+1} on X

$Z_i|y_i = 0 \sim$ Truncated Normal $(X_i^T\beta^t, 1)$, $Z_i \le 0$

$Z_i|y_i = 1 \sim$ Truncated Normal $(X_i^T\beta^t, 1)$, $Z_i > 0$

Since for $W \sim TN(\mu, \sigma^2; (I, \infty))$,

$E(W) = \mu + \sigma \frac{\phi((I-\mu)/\sigma)}{1 - \Phi(\frac{I-\mu}{\sigma})}$

For $W \sim TN(\mu, \sigma^2, (-\infty, I))$,

$E(W) = \mu - \sigma \frac{\phi((I-\mu)/\sigma)}{\Phi(\frac{I-\mu}{\sigma})}$

So $Z_i^{t+1} = \begin{cases} X_i^T\beta^t - \frac{\phi(X_i^T\beta^t)}{\Phi(-X_i^T\beta^t)} & \text{if } y_i = 0 \\ X_i^T\beta^t + \phi(X_i^T\beta^t)/1 - \Phi(-X_i^T\beta^t) & \text{if } y_i = 1 \end{cases}$

(b) Considering starting values of β, we can first set $\beta_1 = ... = \beta_n$ as zero, and then we can set β_0, the intercept of the regression, as $E(I(\hat{Y})) = P(\hat{Y} = 1) = \Phi(\beta_0)$, so $\beta_0 = \Phi^{-1}(\bar{Y})$

(c) First, we generate X from unif(0,1), and set β_1 arbitrarily, set β_0 as 0.5, and $\beta_2 = \beta_3 = 0$. Therefore, we can get our values of Y since $Y \sim \text{Ber}(\Phi(X^T\beta))$. Then, by going through the glm process and calculating $\hat{\beta}/\text{se}(\hat{\beta})$, we find that the ratio is approximately 2 when $beta_1$ takes value 0.8. Then we use these values as starting points for βs and test our EM function. The estimates for the EM function is

approximately the same as the glm estimates.

```
library(Rlab)
```

```
## Rlab 2.15.1 attached.
```

```
##
## Attaching package: 'Rlab'
```

```
## The following objects are masked from 'package:stats':
##
##     dexp, dgamma, dweibull, pexp, pgamma, pweibull, qexp, qgamma,
##     qweibull, rexp, rgamma, rweibull
```

```
## The following object is masked from 'package:datasets':
##
##     precip
```

```
set.seed(2)
n <- 100
#generate X matrix
X_1 <- matrix(runif(3*n), n, 3)
X_0 <- as.matrix(rep(1,n))
X <- cbind(X_0, X_1)
#initialize betas
beta <- matrix(c(0.5, 0.8, 0, 0))
#calculate Pi's
P <- pnorm(X %*% beta)
set.seed(0)
#generate Y vector using Pi
Y <- rbern(n, P)
#Form the glm test
X_df <- cbind.data.frame(X, Y)
names(X_df) <- c("intercept", "X1", "X2", "X3","Y")
result <- glm(Y ~ X1 + X2 + X3, family=binomial(link="probit"),  X_df)
summary(result)
```

```
##
## Call:
## glm(formula = Y ~ X1 + X2 + X3, family = binomial(link = "probit"),
##     data = X_df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2845   0.3914   0.5571   0.7435   0.9893
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.5861     0.4397   1.333   0.1825
## X1            1.1009     0.5130   2.146   0.0319 *
## X2           -0.3516     0.4729  -0.743   0.4572
## X3           -0.1981     0.4983  -0.398   0.6909
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 102.791  on 99  degrees of freedom
## Residual deviance:  97.074  on 96  degrees of freedom
## AIC: 105.07
##
## Number of Fisher Scoring iterations: 5
```

```
# set starting points
b_0 <- qnorm(mean(Y))
beta_sp <- matrix(c(b_0, 0, 0, 0))
# the EM algorithm function
EM <- function(beta, X, Y, eps, max_itr){
  mu = X %*% beta
  i = 0
  cvg = FALSE
  while ((!cvg) & (i < max_itr)) {
    beta_0 <- beta      #save the original beta for comparison
    z_upd = ifelse( Y == 1, mu + dnorm(mu) / (1-pnorm(-mu)), mu - dnorm(mu) / pnorm(-mu)) #update on Z
    beta = solve(t(X) %*% X) %*% t(X) %*% z_upd  #update on beta
    cvg = max(abs(beta - beta_0)) <= eps  #set condition for convergence
    mu = X %*% beta
    i = i + 1
  }
  return(list(beta = t(beta), iterations = i, epsilon = max(abs(beta - beta_0)), convergence = cvg))
}
# test for the result
EM_result = EM(beta_sp, X, Y, eps = 0.0001, max_itr = 100)
EM_result
```

```
## $beta
##          [,1]     [,2]       [,3]       [,4]
## [1,] 0.586148 1.100726 -0.3515283 -0.1981053
##
## $iterations
## [1] 19
##
## $epsilon
## [1] 7.398743e-05
##
## $convergence
## [1] TRUE
```

(d) Using optim() with the BFGS option, we can see that the result for the estimates is approximately the same. The EM algorithm with accuracy of 10^{-8} needed 40 iterations, while the optim() function with the same accuracy needed 9 iterations. Therefore, the optim function required less iterations.

```
EM_result_d = EM(beta_sp, X, Y, eps = 0.00000001, max_itr = 100)
EM_result_d
```

```
## $beta
##           [,1]     [,2]       [,3]       [,4]
## [1,] 0.5861378 1.100861 -0.3515774 -0.1980961
##
## $iterations
## [1] 40
##
## $epsilon
```

```
## [1] 7.501225e-09
##
## $convergence
## [1] TRUE
```

```
MLE = function(beta, X, Y){
  mu = X %*% beta
  negloglike = -sum(Y*pnorm(mu, log.p=T) + (1-Y)*pnorm(-mu, log.p=T))
  return(negloglike)
}

MLE_result = optim(beta_sp, MLE, X=X, Y=Y, method = "BFGS", control = list(trace = TRUE))
```

```
## initial  value 51.395667
## final  value 48.537146
## converged
```

```
MLE_result
```

```
## $par
##             [,1]
## [1,]  0.5861379
## [2,]  1.1008615
## [3,] -0.3515760
## [4,] -0.1980988
##
## $value
## [1] 48.53715
##
## $counts
## function gradient
##       26        9
##
## $convergence
## [1] 0
##
## $message
## NULL
```