PS8

ShubeiWang

11/25/2018

1

(a)

Estimate the mean ϕ

$$E_f(x) = \int xf(x)dx = \int x\frac{f(x)}{q(x)}q(x)dx \approx \frac{1}{m}\Sigma_{i=1}^{m} x_i \frac{f(x_i)}{q(x_i)}$$

Estimate the variance of ϕ

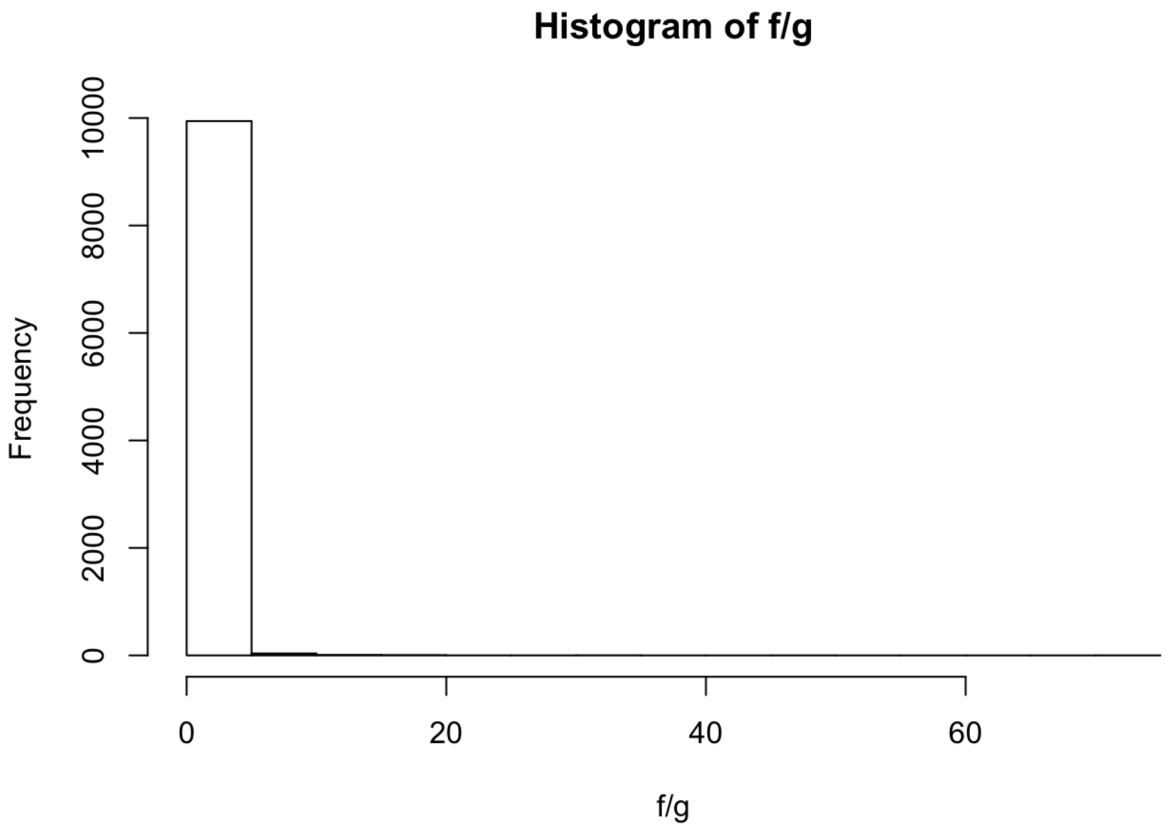
$$\hat{\sigma} = \frac{1}{m}Var(x\frac{f(x)}{q(x)})$$

We can see from the histgram that the weights are distributed close to 0. But the range is rather large which indicates extreme value. That would have a strong influence on the variance.

```
## use importance sampling to estimate the mean

set.seed(0)
m <- 10000
sample <- rnorm(m,-4)
# convert the samples values greater than -4
sample[sample>-4] <- -8-sample[sample>-4]
# estimate the mean
f <- dt(sample,3)/pt(-4,3)
g <- 2*dnorm(sample,-4)
E <- 1/m*sum(sample*f/g)

# create histograms of the weights
hist(f/g)
```




```
# estimate the variance
var <- sum((sample*f/g-E)^2)/m^2

# report the estimates of mean and variance
E
```

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

```
var
```

```
## [1] 0.008995165
```

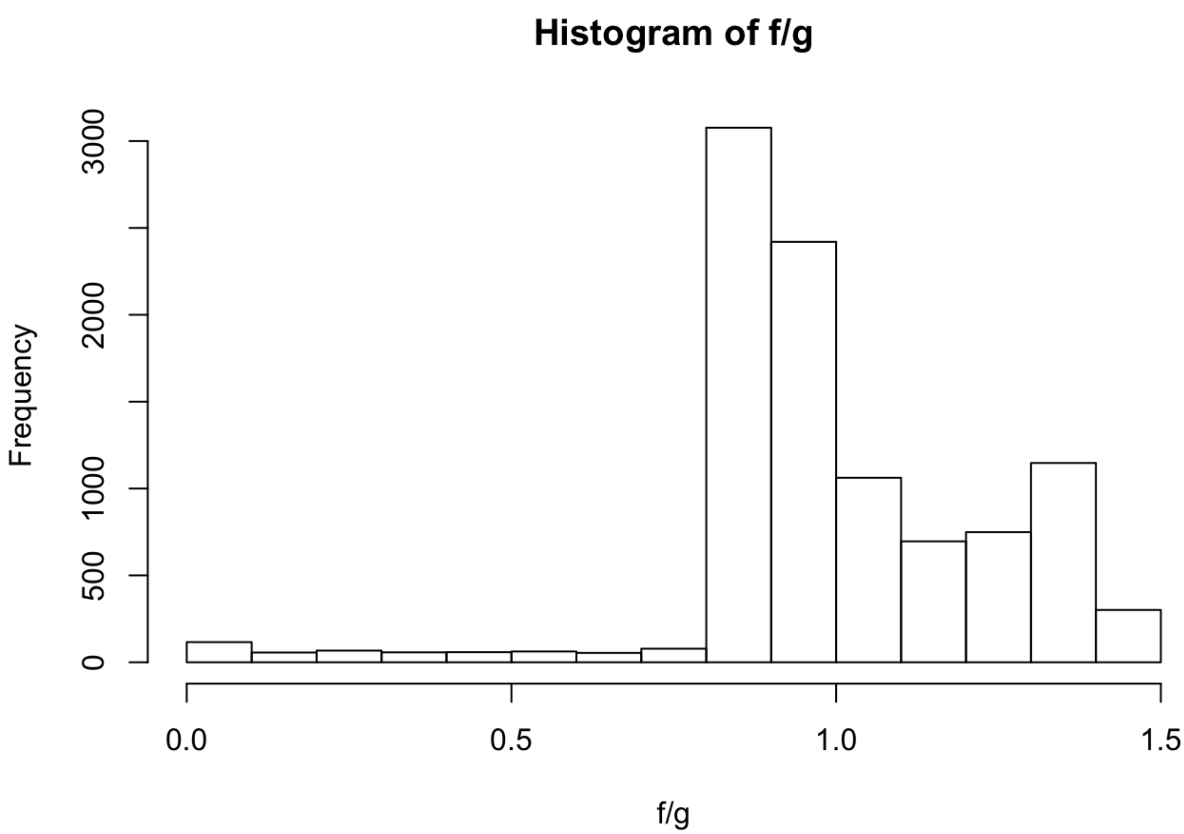
(b)

We can see from the histgram that the weights are distributed between [0,1.5] so it may have a small variance.

```r
## use t distribution as sampling density

set.seed(0)
m <- 10000
sample <- rt(m, df = 1)
sample <- sample-4
# convert the samples values greater than -4
sample[sample>-4] <- -8-sample[sample>-4]
# estimate the mean
f <- dt(sample,3)/pt(-4,3)
g <- 2*dt(sample+4, df = 1)
E <- 1/m*sum(sample*f/g)

# create histograms of the weights
hist(f/g)
```




```r
# estimate the variance
var <- var(sample*f/g)/m

# report the estimates of mean and variance
E
```

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

```r
var
```

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

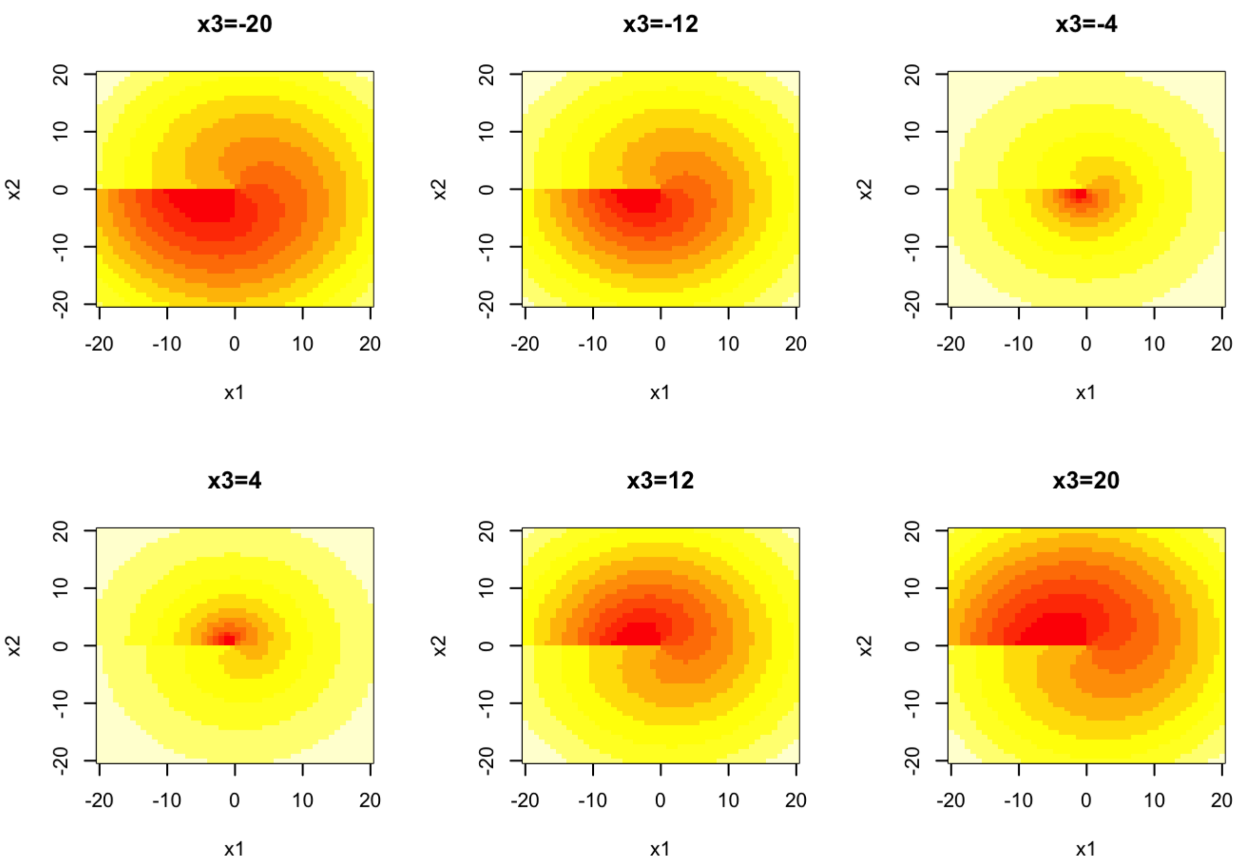
2

```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)
}

## plot slices of the function

x1 <- seq(-20,20,length.out = 50)
x2 <- seq(-20,20,length.out = 50)
par(mfrow = c(2,3))
for(x3 in seq(-20,20,length = 6)){
  obj <- apply(as.matrix(expand.grid(x1,x2)), 1, function(x) f(c(x,x3)))
  image(x1,x2,matrix(log10(obj),50,50),main=paste0("x3=",round(x3,digits = 2)))
}
```



We can see from the results that the optimization converges to $x = (1, 0, 0)$ with $f(x) = 0$. Since$f(x) \geq 0$, we have found the global minimum.

```
set.seed(1)

# set starting point
init <- runif(3,-10,10)
optim(init,f)
```

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

```
nlm(f,p=init)
```

```
## $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
```

3

(a)

(a)

$$Q(\beta|\beta^{(t)}) = E_{z|Y,\beta^{(t)}}[\log f(Y,Z|\beta)|Y,\beta^{(t)}]$$

$$\propto E[-\frac{1}{2}\sum_{i=1}^{n}(z_i - X_i^T\beta)^2 | y, \beta^{(t)}] \overset{\Delta}{=} -\frac{1}{2}E[(Z-X\beta)^T(Z-X\beta)| y, \beta^{(t)}]$$

$\because z_i | y_i = 0, \beta^{(t)} \sim$ truncated Normal $(X_i^T\beta^{(t)}, 1; (-\infty, 0])$

$z_i | y_i = 1, \beta^{(t)} \sim$ truncated normal $(X_i^T\beta^{(t)}, 1; [0, \infty))$

$$\therefore \frac{dQ}{d\beta} = E[Z|y,\beta^{(t)}]^T X - X^TX\beta$$

to maximize $Q \Rightarrow \quad E[Z|y,\beta^{(t)}]^T X = X^TX\beta$

denote $\quad Z^{(t+1)} = E[Z|y,\beta^{(t)}]$

$$\Rightarrow \beta^{(t+1)} = (X^TX)^{-1}X^T Z^{(t+1)}$$

To compute $E[Z|y,\beta^{(t)}]$ we can use the properties in (i)

$$Z_i^{(t+1)} = E[Z|y,\beta^{(t)}]_i = \begin{cases} X_i^T\beta^{(t)} - \dfrac{\phi(X_i^T\beta^{(t)})}{\Phi(-X_i^T\beta^{(t)})} & \text{if } y_i = 0 \\ X_i^T\beta^{(t)} + \dfrac{\phi(X_i^T\beta^{(t)})}{1-\Phi(-X_i^T\beta^{(t)})} & \text{if } y_i = 1 \end{cases}$$

The EM algorithm is as follows:

① select starting value $\beta(0)$ and set $t=0$

② E-step: compute $Z^{(t+1)}$

③ M-step:

③ M-step:
compute $\beta^{(t+1)}$

④ if $\frac{\|\beta^{(t+1)}-\beta^{(t)}\|}{\|\beta^{(t)}\|} < \varepsilon$ then declare $\hat{\beta}=\beta^{(t+1)}$

else : $t \to t+1$, return to step ②.

ε is the tolerance.

(b)

We can choose all parameters to be 0 as starting value.

(c)

```r
n <- 100
## generate random X
set.seed(3)
X <- cbind(1,matrix(runif(3*n),n,3))
colnames(X) <- c("x0","x1","x2","x3")

## choose proper beta
choose_beta <- function(beta0, beta1){

  beta2=0
  beta3=0
  beta = c(beta0,beta1,beta2,beta3)

  # generate observed data
  X_beta = X*beta
  Y <- rbinom(n,1,pnorm(rowSums(X_beta)))

  # perform lm
  data <- cbind.data.frame(X,Y)
  lm <- glm(Y~x1+x2+x3, family = binomial(link = "probit"),data)

  # check beta1/se(beta1)
  q <- lm$coefficients[2]/coef(summary(lm))[,"Std. Error"][2]

  list <- list("beta"=beta, "beta1/se(beta1)"=q, "Y"=Y)
  return(list)
}

# set.seed(3)
# choose_beta(1,0.1)
# $`beta1/se(beta1)` = 2.178989

set.seed(3)
beta <- choose_beta(1,0.1)["beta"]
Y <- choose_beta(1,0.1)["Y"][[1]]

## EM algorithm
probit <- function(X,Y,beta_t){

  iter <- 0

  # E-step
  Estep<- function(X,Y,beta_t){
    Z <- X%*%beta_t
    numerator <- dnorm(Z)*ifelse(Y==1,+1,-1)
    denomenator <- ifelse(Y==1,1-pnorm(-Z),pnorm(-Z))
    Z_new <- Z+numerator/denomenator
    return(Z_new)
  }

  # M-step
  beta_new <- solve(t(X)%*%X)%*%t(X)%*%matrix(Estep(X,Y,beta_t),ncol = 1)

  convergence <- function(new,old){
    if(sum((new-old)^2)/sum(old^2)<1e-10)
      return(TRUE)
```

```
    else
      return(FALSE)
  }

  # follow the procedure until converged
  while(!convergence(beta_new,beta_t)){
    iter <- iter+1
    beta_t <- beta_new
    Z <- Estep(X,Y,beta_t)
    beta_new <- solve(t(X)%*%X)%*%t(X)%*%matrix(Estep(X,Y,beta_t),ncol = 1)
  }

  list <- list("beta"=beta_new, "iter"=iter)
    return(list)
}

## test the function
beta_t <- c(1,0.1,0,0)
probit(X,Y,beta_t)
```

```
## $beta
##          [,1]
## x0 -0.4722844
## x1  0.5770955
## x2  1.7375672
## x3 -0.4453417
##
## $iter
## [1] 14
```

(d)

We can see that optim() and EM give very close results. EM takes 14 iterations and BFGS takes 10.

```
## log-liklihood function
loglik <- function(beta,X,Y){
  p <- pnorm(X%*%beta,lower.tail = TRUE)
  loglik <- sum(Y*log(p)+(1-Y)*log(1-p))
  return(loglik)
}

optim(beta_t, fn = loglik, X=X, Y=Y, method = 'BFGS', control = list(trace = TRUE,maxit=10000
, fnscale=-1))
```

```
## initial  value 77.202794
## iter  10 value 58.597733
## iter  10 value 58.597733
## final  value 58.597733
## converged
```

```
## $par
## [1] -0.4723020  0.5771105  1.7375843 -0.4453361
##
## $value
## [1] -58.59773
##
## $counts
## function gradient
##       19       10
##
## $convergence
## [1] 0
##
## $message
## NULL
```