Stat243 ps8

Winnie Gao

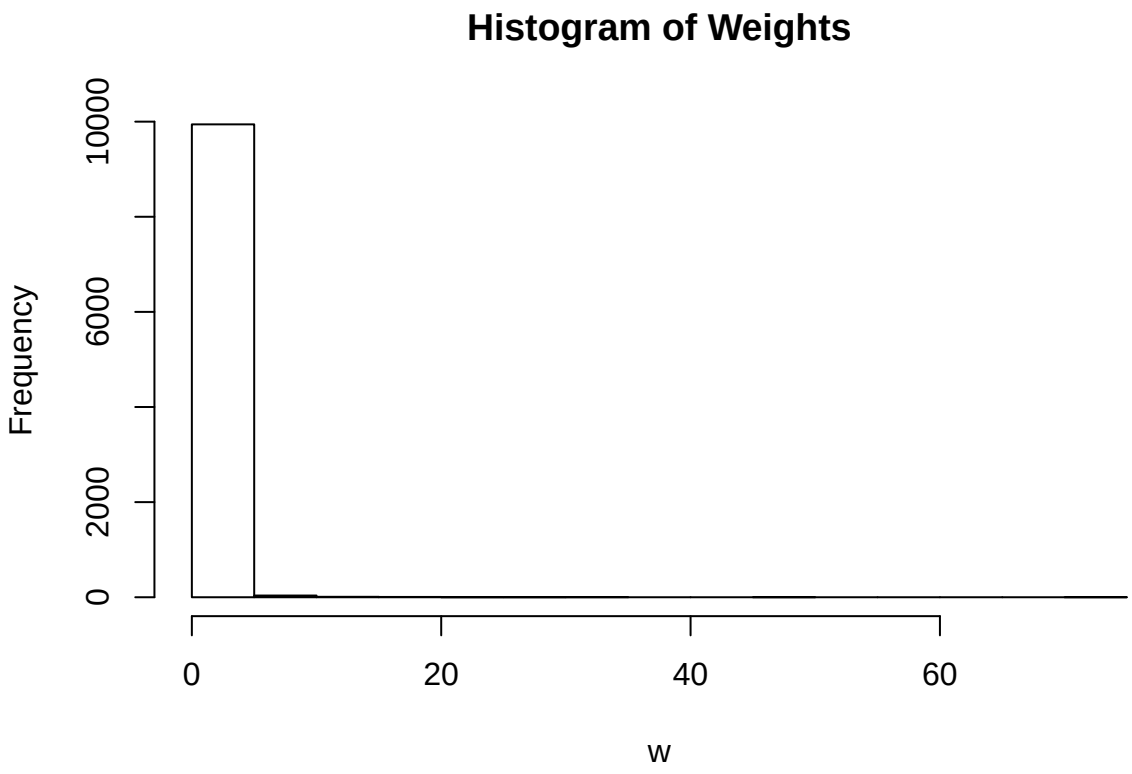
11/26/2018

Problem 1

(a) Use importance sampling to estimate the mean of a truncated t distribution with 3 degrees of freedom, truncated such that X < (-4). Have your sampling density be a normal distribution centered at -4 and then truncated so you only sample values less than -4 (this is called a half-normal distribution). You should be able to do this without discarding any samples (how?). Use m = 10000 samples. Create histograms of the weights f(x)/g(x) to get a sense for whether Var($\hat{\phi}$) is large. Note if there are any extreme weights that would have a very strong influence on $\hat{\phi}$. Estimate Var($\hat{\phi}$). Hint: remember that your f(x) needs to be appropriately normalized or you need to adjust the weights per the class notes.

```
set.seed(0)
n <- 10000

x <- -abs(rnorm(n))-4 #samples from half-normal distribution centered at -4
f <- dt(x, df = 3)/pt(-4, df=3) #density of x under f, a t dist with df=3, truncated at -4
g <- dnorm(x+4)/pnorm(0) #density of x under g, a half-normal dist centered at -4
w <- f/g  #weights
hist(x = w, main = 'Histogram of Weights')
```




```
x_star <- x*w #samples
mean((x_star-(-4))^2) #estimated variance
```

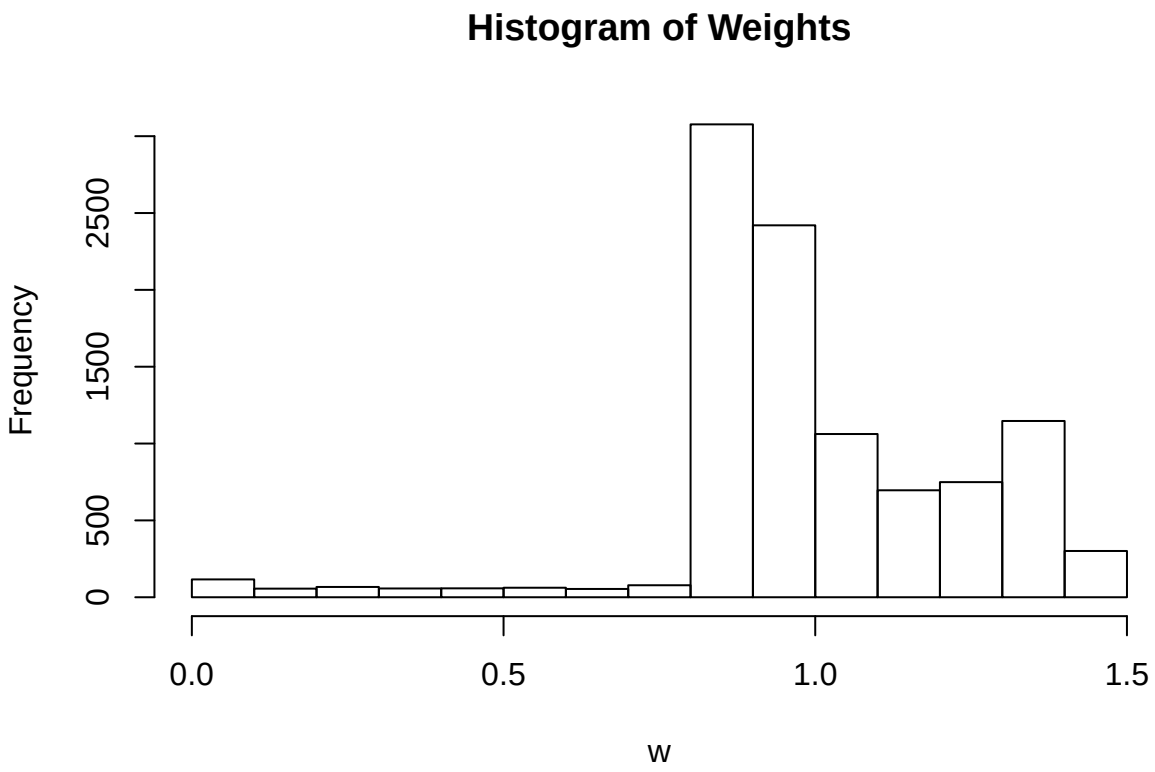
```
## [1] 90.01221
```

Ans: According to the histogram of weights, we can see only a really small portion of weights are greater than 5. But there are even weights greater than 60, which would have a very strong influence on $\hat{\phi}$. Estimated $\text{Var}(\hat{\phi})$ is 90.01221.

(b) Now use importance sampling to estimate the mean of the same truncated t distribution with 3 degrees of freedom, truncated such that X < (-4), but have your sampling density be a t distribution, with 1 degree of freedom (not 3), centered at -4 and truncated so you only sample values less than -4. Again you shouldn't have to discard any samples. Respond to the same questions as above in part (a).

```
set.seed(0)
n <- 10000

x <- -abs(rt(n, df = 1))-4 #samples from t dist with df=1 truncated at -4
f <- dt(x, df = 3)/pt(-4, df=3)#density of x under f, a t dist with df=3, truncated at -4
g <- dt(x+4, df = 1)/pt(0, df = 1) #density of x under g, a half-normal dist centered at -4
w <- f/g  #weights
hist(x = w, main = 'Histogram of Weights')
```




```
x_star <- x*w #samples
mean((x_star-(-4))^2) #estimated variance
```

```
## [1] 14.12048
```

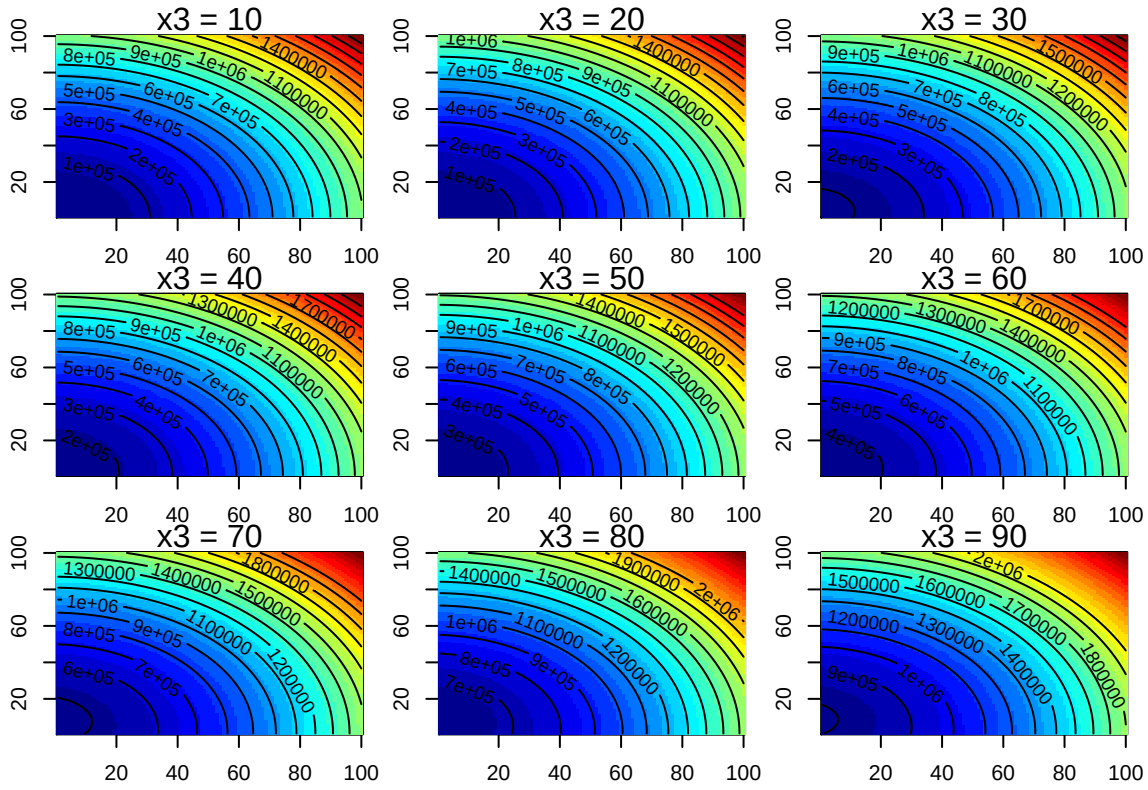
Ans: According to the histogram of weights, we can see all of the weights fall into the range between 0 and 1.5. Therefore there are no extreme weights. So the estimated variance should be smaller compared with normal approximation. Estimated $\text{Var}(\hat{\phi})$ is 14.12048.

Problem 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.

```
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)
}
```

```r
library(fields)
par(mfrow = c(3,3), mar = c(2, 2, 1, 1), oma = c(3, 3, 0.5, 1))
result <- matrix(0, 100, 100)
x <- seq(1, 100, 1)
x3_ls = seq(10,90,10)
for (m in x3_ls){
  for(i in x){
    for(j in x){
      result[i,j]=f(c(i,j,m))
    }
  }
  image(x, x, result, col = tim.colors(32))
  contour(x, x, result, levels = seq(100000, 2000000, by=100000), add = TRUE, col = 'black')
  mtext(paste0('x3 = ', m), side = 3)
}
```




```r
x_init <- matrix(c(c(0,0,0),c(-10,10,10),c(10,10,10)),3,3, byrow = TRUE)
for (i in 1:3){
  print(paste0('Starting point of x are: ', paste0(x_init[i,], collapse = ',')))
  optim_res <- optim(par = x_init[i,], f)
  print(paste0('Optim: x values that minimize the function f are ',
               paste0(optim_res$par, collapse = ',')))
  print(paste0('Optim: minimum of the function f is ', optim_res$value))
  nlm_res <- nlm(f, p = x_init[i,])
  print(paste0('Nlm: x values that minimize the function f are ',
               paste0(nlm_res$estimate, collapse = ',')))
```

```
  print(paste0('Nlm: minimum of the function f is ', nlm_res$minimum))
  print('------------------------------')
}
```

```
## [1] "Starting point of x are: 0,0,0"
## [1] "Optim: x values that minimize the function f are 0.999978292008071,0.00273069833992297,0.004284
## [1] "Optim: minimum of the function f is 1.87685068944381e-05"
## [1] "Nlm: x values that minimize the function f are 0,0,0"
## [1] "Nlm: minimum of the function f is 100"
## [1] "------------------------------"
## [1] "Starting point of x are: -10,10,10"
## [1] "Optim: x values that minimize the function f are 1.00073270422073,-0.00331723117909967,-0.00534
## [1] "Optim: minimum of the function f is 8.35284692190126e-05"
## [1] "Nlm: x values that minimize the function f are 1.00000000056058,-4.75202215791134e-10,4.7403531
## [1] "Nlm: minimum of the function f is 1.83024650175485e-16"
## [1] "------------------------------"
## [1] "Starting point of x are: 10,10,10"
## [1] "Optim: x values that minimize the function f are 1.0000509522865,-0.00906176203014949,-0.014139
## [1] "Optim: minimum of the function f is 0.000208707787236223"
## [1] "Nlm: x values that minimize the function f are 1.00000000001286,2.41858536900428e-10,3.4722989
## [1] "Nlm: minimum of the function f is 2.79245070524621e-19"
## [1] "------------------------------"
```

Ans: optim and nlm will provide different results with same starting points. Different starting points will also provide different minima.

Problem 3

Consider probit regression, which is an alternative to logistic regression for binary outcomes. The probit model is $Y_i \sim Ber(p_i)$ for $p_i = P(Y_i = 1) = \Phi({X_i}^T\beta)$ where Φ is the standard normal CDF. We can rewrite this model with latent variables, one latent variable, z_i, for each observation:

$$y_i = I(z_i > 0)$$

$$z_i \sim N({X_i}^T\beta, 1)$$

(a) an EM algorithm to estimate β, taking the complete data to be fY;Zg. You'll need to make use the the mean and variance of truncated normal distributions (see hint below). Be careful that you carefully distinguish β from the current value at iteration t, β^t, in writing out the expected log-likelihood and computing the expectation and that your maximization be with respect to β (not β^t). Also be careful that your calculations respect the fact that for each z_i you know that it is either bigger or smaller than 0 based on its yi. You should be able to analytically maximize the expected log likelihood.

Ans:

$$ll(\beta) = -\frac{nlog(2\pi)}{2} - \frac{\sum_{i=1}^{n}(z_i - {X_i}^T\beta)^2}{2}$$

E-step:

$$Q(\beta|\beta^t) = E[ll(\beta)|y_i, \beta^t] = -\frac{nlog(2\pi)}{2} - \frac{\sum_{i=1}^{n} E({z_i}^2|y_i, \beta^t) - {X_i}^T\beta * E(z_i|y_i, \beta^t) + {X_i}^T\beta\beta^T X_i}{2}$$

$$= -\frac{nlog(2\pi)}{2} - \frac{\sum_{i=1}^{n} Var(z_i|y_i, \beta^t)}{2} - \frac{\sum_{i=1}^{n}[E(z_i|y_i, \beta^t)]^2 - {X_i}^T\beta * E(z_i|y_i, \beta^t) + {X_i}^T\beta\beta^T X_i}{2}$$

M-step:

$$\frac{\partial Q(\beta|\beta^t)}{\partial \beta} = \sum_{i=1}^{n} {X_i}^T E(z_i|y_i, \beta^t) - {X_i}^T X_i \beta = 0$$

$$Update\ \beta^{t+1} = \frac{\sum_{i=1}^{n} {X_i}^T E(z_i|y_i, \beta^t)}{\sum_{i=1}^{n} {X_i}^T X_i}$$

$$E(z_i|y_i, \beta^t) = \begin{cases} {X_i}^T\beta + \frac{\phi(-{X_i}^T\beta)}{1-\Phi(-{X_i}^T\beta)}, & y_i = 1 \\ {X_i}^T\beta - \frac{\phi(-{X_i}^T\beta)}{\Phi(-{X_i}^T\beta)}, & y_i = 0 \end{cases}$$

(b) Propose reasonable starting values for β.

```
guess <- lm(y~x1+x2+x3)
start_beta <- guess$coefficients[1:4]
```

Ans: We can use linear regression to propose reasonable starting values for β.

(c) Write an R function, with auxiliary functions as needed, to estimate the parameters. Make use of the initialization from part (b). You may use lm() for the update steps. You'll need to include criteria for deciding when to stop the optimization. Test your function using data simulated from the model, with say β_0; β_1; β_2; β_3. Take n = 100 and the parameters such that $\hat{\beta}_1/se(\hat{\beta}_1) \approx 2$ *and* $\beta_2 = \beta_3 = 0$. (In other words, I want you to choose β_1 such that the signal to noise ratio in the relationship between x_1 and y is moderately large.)

```
probEM<-function(y, x1, x2, x3, start_beta=c(1,0.5,0,0),tolerance = .Machine$double.eps, maxN = 10000){
  n <- length(y)
  beta0 <- start_beta[1]
  beta1 <- start_beta[2]
  beta2 <- start_beta[3]
  beta3 <- start_beta[4]

  counter <- 1
  diff <- Inf

  while(diff > tolerance & counter < maxN){
    st_0 <- beta0
    st_1 <- beta1
    st_2 <- beta2
    st_3 <- beta3

    #E step
    mu <- beta0 + beta1*x1 + beta2*x2 + beta3*x3

    z <- ifelse(y==1,
                mu+dnorm(mu,mean=0, sd=1)/pnorm(mu, mean=0, sd=1),
                mu-dnorm(mu,mean=0, sd=1)/pnorm(-mu, mean=0, sd=1))
    #M step
    lreg <- lm(z~x1+x2+x3)
    beta0 <- lreg$coefficients[1]
    beta1 <- lreg$coefficients[2]
    beta2 <- lreg$coefficients[3]
```

```
    beta3 <- lreg$coefficients[4]

    diff <- sum(abs(c(beta0-st_0,beta1-st_1,beta2-st_2,beta3-st_3)))/sum(abs(c(st_0,st_1,st_2,st_3)))

    counter <- counter + 1
  }
  return(c(beta0, beta1, beta2, beta3, counter))
}

#test if we find appropriate betas
check <- function(beta0, beta1){
  set.seed(0)
  n<-100
  beta2<-0
  beta3<-0
  x1<-rnorm(n)
  x2<-rnorm(n)
  x3<-rnorm(n)

  mu <- beta0 + beta1*x1 + beta2*x2 + beta3*x3
  y <- rbinom(n, 1, prob = pnorm(mu))
  summary(glm(y~x1+x2+x3, family = binomial(link = 'probit')))$coef[2,3]
}

ts1 <- seq(0.1,1,0.1)
ts2 <- seq(0.1,1,0.1)
checkcase <- matrix(0,10,10)
for (i in 1:10){
  for (j in 1:10){
    checkcase[i,j] = check(ts1[i],ts2[j])
  }
}

#After taking a look at checkcase, I found the case that is closest to 2
check(ts1[1],ts2[5])
```

```
## [1] 1.946456
```

```
print(paste0('The values of beta0 and beta1 I select are ', ts1[1], ', ', ts2[5]))
```

```
## [1] "The values of beta0 and beta1 I select are 0.1, 0.5"
```

Ans: Therefore I choose β_0 to be 0.1 and β_1 to be 0.5 in order to meet the restrictions.

```
set.seed(0)
n<-100
beta0 <- 0.1
beta1 <- 0.5
beta2 <- 0
beta3 <- 0

x1 <- rnorm(n)
x2 <- rnorm(n)
x3 <- rnorm(n)
```

```r
mu <- beta0 + beta1*x1 + beta2*x2 + beta3*x3
set.seed(0)
y <- rbinom(n, 1, prob = pnorm(mu))

lreg <- lm(y~x1+x2+x3)
start_beta <- as.double(coef(lreg))
probEM(y,x1,x2,x3,start_beta = start_beta)
```

```
## (Intercept)          x1          x2          x3
##   0.1767602   0.5761596  -0.1629955   0.0243073  58.0000000
```

(d) A different approach to this problem just directly maximizes the log-likelihood of the observed data. Estimate the parameters (and standard errors) for your test cases using optim() with the BFGS option in R. Compare how many iterations EM and BFGS take.

```r
loglik <- function(beta,X,y){
  p <- pnorm(X%*%beta, mean=0, sd=1, lower.tail = TRUE, log.p = FALSE)
  logl <- sum(y*log(p)+(1-y)*log(1-p))
  return(logl)
}

#initial setting
X <- cbind(1, x1, x2, x3)
#fnscale -- turn into maximization problem, trace -- show tracing info
opt_res <- optim(start_beta, fn = loglik, X = X, y = y, method = 'BFGS',
                 control = list(trace = TRUE, maxit = 10000, fnscale = -1), hessian = TRUE)
```

```
## initial  value 68.844638
## final  value 61.418050
## converged
```

```r
opt_res
```

```
## $par
## [1]  0.17676025  0.57615970 -0.16299554  0.02430732
##
## $value
## [1] -61.41805
##
## $counts
## function gradient
##       27        9
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##            [,1]       [,2]        [,3]        [,4]
## [1,] -57.591504   2.457709   3.5378073  -4.8022404
## [2,]   2.457709 -36.385748  -6.8846588  -1.5743789
## [3,]   3.537807  -6.884659 -53.7405860   0.8981376
```

```
## [4,]  -4.802240  -1.574379   0.8981376 -64.4911206
```

```r
#standard error
print(sqrt((-1)*diag(solve(opt_res$hessian))))
```

```
## [1] 0.1325907 0.1681207 0.1383298 0.1250129
```

Ans: EM uses 58 iterations but BFGS only uses 27 iterations.