STAT 243: Problem Set 8

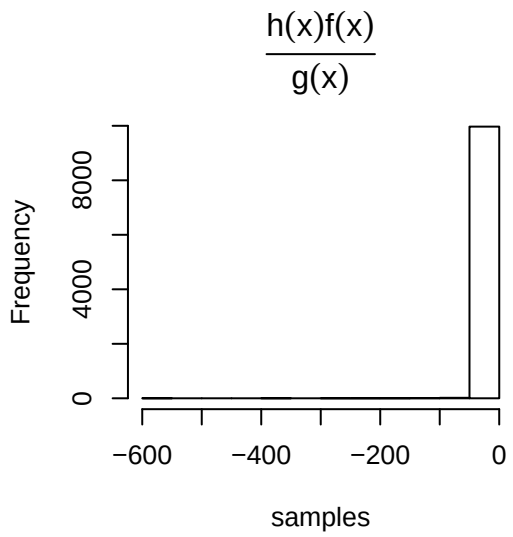
Jin Rou New [jrnew]

December 2, 2014

1 Problem 1

(a) Since we wish to obtain the mean, our $h(x)$ is simply x. To avoid discarding any samples, we can use the absolute values of the samples from a standard normal distribution and subtract 4 from all the absolute values so that the samples are from a half-normal distribution centered at -4. The same can be done for a half-t-distribution.
$Var(\hat{\mu})$ is large; we see from the histogram that the samples of $\hat{\mu}$ range from -600 to 0. Most of the weights are less than 5, but there are some weights greater than that, even up to a maximum of 72. Such high weights give rise to overly influential points.

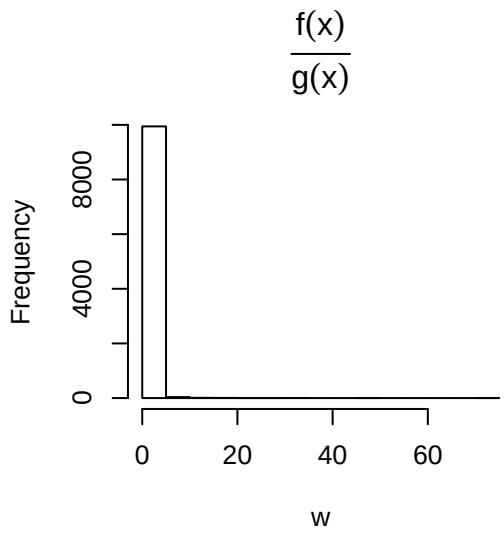
```
set.seed(0)
m <- 10000
x <- - abs(rnorm(m)) - 4 # samples from a half-normal distri centered/trunacted at -4
f <- dt(x, df = 3)/pt(-4, df = 3)  # density of x under f, a t distri with df 3, truncated at -4
g <- dnorm(x + 4)/pnorm(0)  # density of x under g, a half-normal distri centered/truncated at -4
w <- f/g # weights
samples <- x*w
summary(samples)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -558.0    -3.4    -3.0    -4.2    -2.8    -2.8

par(cex = 0.8)
hist(samples, main = expression(frac(h(x)*f(x), g(x))))
```



```
summary(w)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.60    0.61    0.66    0.83    0.75   72.40
```

```
hist(w, main = expression(frac(f(x),g(x))))
```




```
mean(samples) # estimated mean of a t distri with df 3, truncated at -4
```

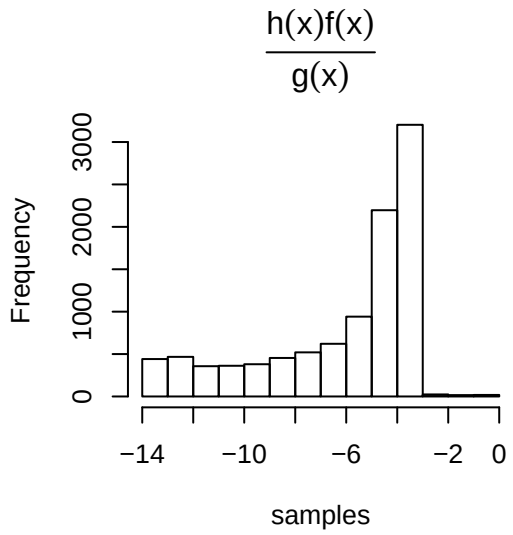
```
## [1] -4.246
```

(b) $Var(\hat{\mu})$ is not large since the samples of $\hat{\mu}$ lie between -14 and 0. The weights lie between 0 and 1.5, so there are no extreme weights. The choice of a t-distribution as opposed to a normal distribution for $g(x)$ gives rise to a better estimate of the desired mean since a t-distribution has heavier tails.

```
set.seed(0)
m <- 10000
x <- - abs(rt(m, df = 1)) - 4 # samples from a t distri centered/truncated at -4
f <- dt(x, df = 3)/pt(-4, df = 3) # density of x under f, a t distri with df 3, truncated at -4
g <- dt(x + 4, df = 1)/pt(0, df = 1)  # density of x under g, a t distri centered/truncated at -4
w <- f/g # weights
samples <- x*w
summary(samples)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -13.400  -7.940  -4.650  -6.210  -3.900  -0.047
```

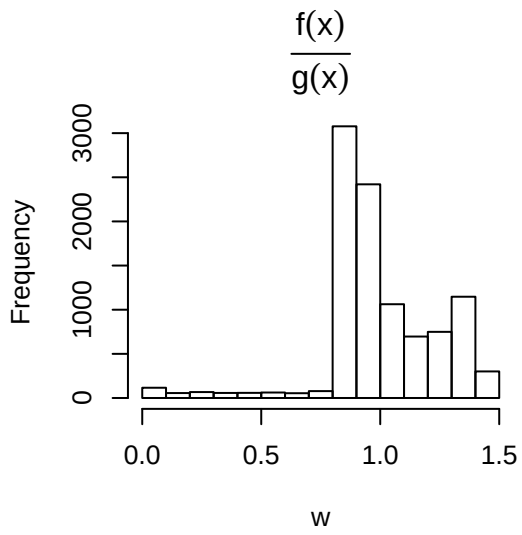
```
par(cex = 0.8)
hist(samples, main = expression(frac(h(x)*f(x), g(x))))
```




```
summary(w)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   0.875   0.948   0.999   1.160   1.410
```

```
hist(w, main = expression(frac(f(x),g(x))))
```



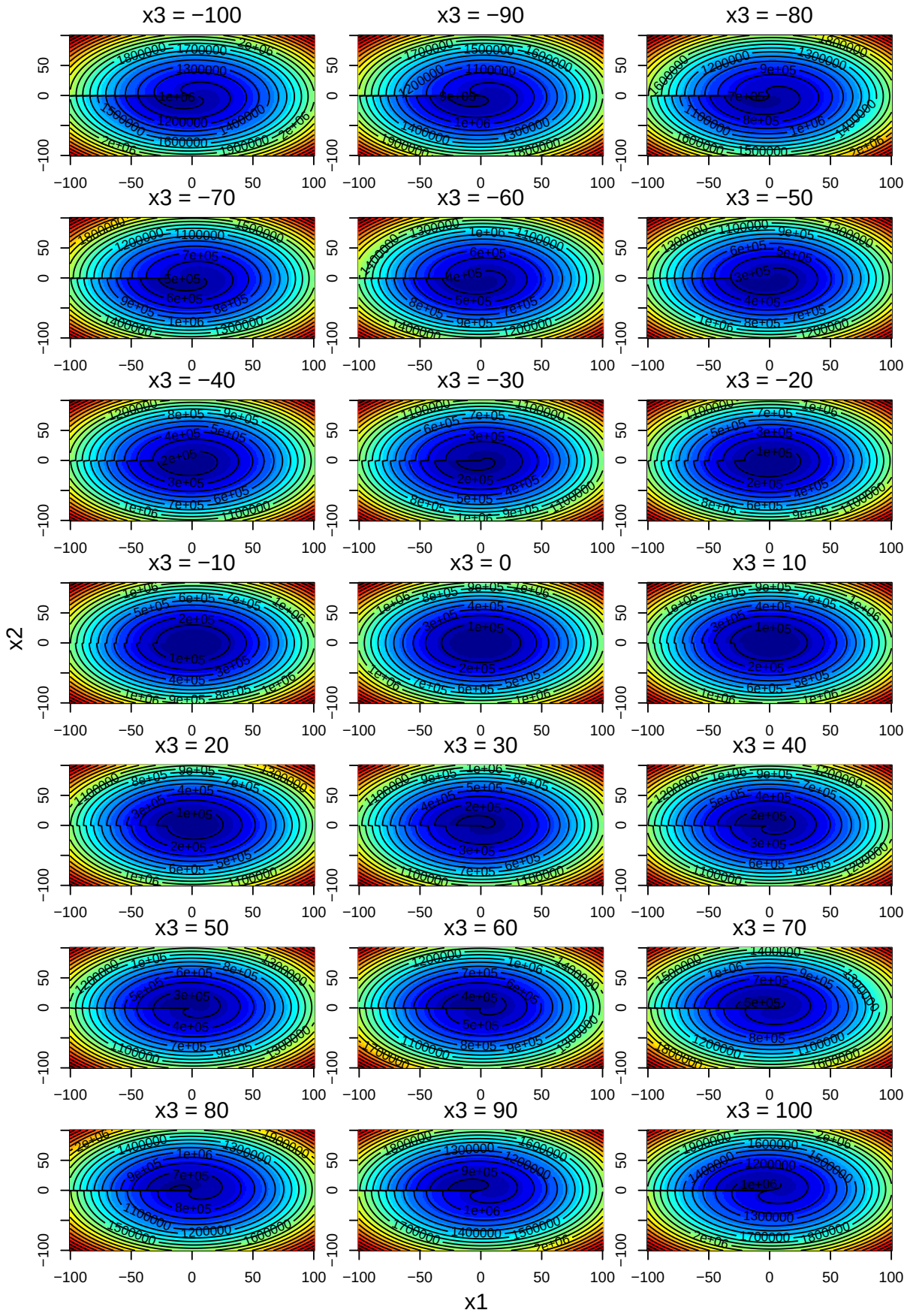

```
mean(samples) # estimated mean of a t distri with df 3, truncated at -4
```

```
## [1] -6.209
```

2 Problem 2

`optim` and `nlm` give different results even with the same starting points. There are multiple local minima since different starting points give different minima.

```r
theta <- function(x1, x2) atan2(x2, x1)/(2*pi)
f <- function(x1, x2, x3) {
  f1 <- 10*(x3 - 10*theta(x1, x2))
  f2 <- 10*(sqrt(x1^2+x2^2)-1)
  f3 <- x3
  return(f1^2+f2^2+f3^2)
}

library(fields)
x <- seq(-100, 100, 1)
par(mfrow = c(7, 3), mar = c(1.5, 1.5, 2, 1), oma = c(3, 3, 0.5, 1))
X_all <- expand.grid(x, x, x)
result_all <- f(X_all[, 1], X_all[, 2], X_all[, 3])
for (x3 in x[seq(1, length(x), 10)]) {
  result <- outer(x, x, function(x1, x2) f(x1, x2, x3))
  image(x, x, result, col = tim.colors(32))
  contour(x, x, result, levels = seq(min(result_all), max(result_all), by = 100000),
          add = TRUE, col = "black")
  mtext(paste("x3 =", x3), side = 3, line = 0.5)
}
mtext("x1", side = 1, line = 1, outer = TRUE)
mtext("x2", side = 2, line = 1, outer = TRUE)
```


x3 = −100
x3 = −90
x3 = −80
x3 = −70
x3 = −60
x3 = −50
x3 = −40
x3 = −30
x3 = −20
x3 = −10
x3 = 0
x3 = 10
x3 = 20
x3 = 30
x3 = 40
x3 = 50
x3 = 60
x3 = 70
x3 = 80
x3 = 90
x3 = 100
x1
x2

```r
fnew <- function(x) f(x[1], x[2], x[3])
X_init <- matrix(c(c(0, 0, 0),
                   c(-50, -50, -50),
                   c(50, 50, 50)), 3, 3, byrow = TRUE)
for (i in 1:nrow(X_init)) {
  cat("Starting values of x are:", paste(X_init[i, ], collapse = ", "), "\n")
  res <- optim(par = X_init[i, ], fnew)
  cat("Using optim...\n")
  cat("Values of x are that minimize the function:\n", paste(res$par, collapse = ", "), "\n")
  cat("Minimum of the function:", res$value, "\n")
  res2 <- nlm(fnew, p =  X_init[i, ])
  cat("Using nlm\n")
  cat("Values of x are that minimize the function:\n", paste(res2$estimate, collapse = ", "), "\n")
  cat("Minimum of the function:", res2$minimum, "\n")
  cat("-------------------------------------------------\n")
}
```

```
## Starting values of x are: 0, 0, 0
## Using optim...
## Values of x are that minimize the function:
##  0.999978292008071, 0.00273069833992297, 0.00428463978017101
## Minimum of the function: 1.877e-05
## Using nlm
## Values of x are that minimize the function:
##  0, 0, 0
## Minimum of the function: 100
## -------------------------------------------------
## Starting values of x are: -50, -50, -50
## Using optim...
## Values of x are that minimize the function:
##  0.996283723800005, 0.0458378202725633, 0.0703910496537494
## Minimum of the function: 0.006438
## Using nlm
## Values of x are that minimize the function:
##  1.00000000262574, -2.40278400888622e-09, 2.37351101623547e-09
## Minimum of the function: 4.536e-15
## -------------------------------------------------
## Starting values of x are: 50, 50, 50
## Using optim...
## Values of x are that minimize the function:
##  0.986854200055761, 0.143879401874988, 0.225167763592301
## Minimum of the function: 0.05419
## Using nlm
## Values of x are that minimize the function:
##  0.999999492217191, -8.22574453915421e-05, -0.000130118659223364
## Minimum of the function: 1.702e-08
## -------------------------------------------------
```

3 Problem 3

(c) Once all the quantities such as $Q(\theta|\theta_t)$ and maximum likelihood estimates of the parameters were derived analytically, it was trivial to implement the EM algorithm in R. Starting values were obtained by carrying

out a simple linear regression procedure on the data as if it were not censored. At each iteration, the required quantities as described in part 3(a) are evaluated and updated.
The stopping criteria are 1) convergence of $Q(\theta|\theta_t)$, where convergence is reached when this value at the current iteration is within $1x10^{-10}$ of that of the previous iteration, and 2) the maximum number of iterations (default 10,000) is reached.

```r
# Simulate data
set.seed(1)
n <- 100
x <- rnorm(n)
beta0_true <- 2
beta1_true <- 0.5
sigma_true <- beta1_true*sqrt(sum((x - mean(x))^2))/3 # for signal to noise ratio of 3
ytemp <- beta0_true + beta1_true*x + rnorm(n, 0, sigma_true)
tau1 <- quantile(ytemp, 0.8) # tau for 20% exceedances
tau2 <- quantile(ytemp, 0.2) # tau for 80% exceedances
truncated1 <- ytemp > tau1
truncated2 <- ytemp > tau2
y1 <- pmin(tau1, ytemp)
y2 <- pmin(tau2, ytemp)
data1 <- list(x = x, y = y1, truncated = truncated1, tau = tau1)
data2 <- list(x = x, y = y2, truncated = truncated2, tau = tau2)

# Check signal to noise ratio
mod_temp <- summary(lm(ytemp ~ x))
signal_to_noise_ratio <- mod_temp$coefficients[2, 1]/mod_temp$coefficients[2, 2]
signal_to_noise_ratio
```

```
## [1] 3.106
```

```r
#' Perform the EM algorithm to estimate linear regression parameters for truncated data.
#'
#' @param data List with variables x, y, truncated and tau.
#' @param num_max_iterations Maximum number of iterations before EM algorithm
#' stops if convergence is still not reached.
#' @return List containing results of EM algorithm.
em_algorithm <- function(
  data,
  num_max_iterations = 10000
) {
  # Initialise parameter values
  mod <- summary(lm(data$y ~ data$x))
  beta0_all <- beta0_prev <- mod$coefficients[1, 1]
  beta1_all <- beta1_prev <- mod$coefficients[2, 1]
  sigma_all <- sigma_prev <- mod$sigma*(n-2)/n

  x <- data$x
  xtrunc <- x[data$truncated == 1]
  ystar <- data$y
  q_theta_all <- NA
  q_theta_prev <- 0
  q_theta <- -10000
  i <- 1

  while (abs(q_theta - q_theta_prev) > 1e-10 & i <= num_max_iterations) {
```

```
    if (i > 1) {
      beta0_prev <- beta0
      beta1_prev <- beta1
      sigma_prev <- sigma
      q_theta_prev <- q_theta
    }

    # Compute E(Y_{trunc, i} | theta_t) for i = 1, ..., c
    mu_trunc <- beta0_prev + beta1_prev*xtrunc
    tau_star <- (data$tau - mu_trunc)/sigma_prev
    rho_tau_star <- dnorm(tau_star)/(1 - pnorm(tau_star))
    mean_ytrunc <- mu_trunc + sigma_prev*rho_tau_star
    ystar[data$truncated == 1] <- mean_ytrunc
    var_ytrunc <- (sigma_prev^2)*(1 + tau_star*rho_tau_star - rho_tau_star^2)

    # Regress ystar = {Yobs, E(Y_{trunc, i} | theta_t)} on {x}
    mod <- lm(ystar ~ x)
    beta0 <- mod$coefficients[1]
    beta1 <- mod$coefficients[2]

    # Compute sigma and q_theta
    rss_star <- sum((ystar - beta0 - beta1*x)^2) + sum(var_ytrunc) # corresponds to A + B + C in deriva
    sigma <- sqrt(rss_star/n)
    q_theta <- -(n/2)*log(2*pi*sigma^2) - rss_star/(2*sigma^2)

    # Save current values
    beta0_all <- c(beta0_all, beta0)
    beta1_all <- c(beta1_all, beta1)
    sigma_all <- c(sigma_all, sigma)
    q_theta_all <- c(q_theta_all, q_theta)
    i <- i + 1
  }
  cat(paste0("Results of EM algorithm:\nbeta0: ", tail(beta0_all, 1), "\n",
             "beta1: ", tail(beta1_all, 1), "\n",
             "sigma: ", tail(sigma_all, 1), "\n",
             "Q(theta|theta_t): ", tail(q_theta_all, 1), "\n"))
  return(list(beta0 = beta0_all, beta1 = beta1_all, sigma = sigma_all,
              q_theta = q_theta_all))
}

em1 <- em_algorithm(data = data1)
```

```
## Results of EM algorithm:
## beta0: 1.92534766634096
## beta1: 0.517277573528124
## sigma: 1.38272227346804
## Q(theta|theta_t): -174.299275116462
```

```
em2 <- em_algorithm(data = data2)
```

```
## Results of EM algorithm:
## beta0: 1.58516269644711
## beta1: 0.407096769678552
## sigma: 1.04110157573639
## Q(theta|theta_t): -145.921789323239
```

(d) `optim` minimizes a function, so I inputed the negative log likelihood function as the function to be minimized. Since the parameter σ must be positive, I worked with it on the log scale.
The Hessian evaluated at the MLE when the negative log likelihood is minimized is equivalent to the observed Fisher information evaluated at the MLE. The inverse of the observed Fisher information gives an estimate of the covariance matrix, so the square root of the diagnonal elements give the standard errors of the parameter estimates.

```
# Get initial parameter values
init1 <- c(beta0 = em1$beta0[1], beta1 = em1$beta1[1], log_sigma = log(em1$sigma[1]))
init2 <- c(beta0 = em2$beta0[1], beta1 = em2$beta1[1], log_sigma = log(em2$sigma[1]))

negloglikelihood <- function(par, data) {
  beta0 <- par[1]
  beta1 <- par[2]
  log_sigma <- par[3]
  mu <- beta0 + beta1*x
  # Log likelihood function for censored data
  lik <- sum(dnorm(data$y[!data$truncated], mu[!data$truncated], exp(log_sigma),
                   log = TRUE),
             pnorm(data$tau, mu[data$truncated], exp(log_sigma),
                   log = TRUE, lower.tail = FALSE))
  return(-lik)
}

solve1 <- optim(par = init1, fn = negloglikelihood, data = data1, method = "BFGS", hessian = TRUE)
cov1 <- solve(solve1$hessian) # estimate of covariance matrix
sqrt(diag(cov1)) # standard errors of parameter estimates

##     beta0     beta1 log_sigma
##   0.14311   0.15931   0.08323

solve2 <- optim(par = init2, fn = negloglikelihood, data = data2, method = "BFGS", hessian = TRUE)
cov2 <- solve(solve2$hessian) # estimate of covariance matrix
sqrt(diag(cov2)) # standard errors of parameter estimates

##     beta0     beta1 log_sigma
##    0.2545    0.1733    0.1878
```