1) a)

Since $Var(V^T A V) = E[(V^T A V)^2] - tr(A)^2$

it suffices to minimize $E[(V^T A V)^2] = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} a_{ij} a_{kl} E(V_i V_j V_k V_l)$

Since V_i's are independent and $E(V_i)=0$, $var(V_i)=1$

Then $E(V_i V_j) = E(V_i)E(V_j) = 0$ if $i \neq j$

Then, $E(V_i V_j V_k V_l) \neq 0$ if $i=j=k=l$

if $i=j=k=l$ then we have $\sum_{i=1}^{n} a_{ii}^2 E(V_i^4)$

But there's another case where $E(V_i V_j V_k V_l) \neq 0$, if $i=j \wedge k=l$ OR $i=k \wedge j=l$ OR $i=l \wedge j=k$

This follows from the fact if V_i, V_j are independent $\Rightarrow V_i^2, V_j^2$ are independent.

Then $E(V_i^2 V_k^2) = E(V_i^2)E(V_k^2) = var(V_i)var(V_k) = 1$

Then we have $\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ii}a_{jj} + \sum_{i=1}^{n}\sum_{j=1} a_{ij}a_{ij} + \sum_{i=1}^{n}\sum_{j=1} a_{ij}a_{ji}$

So, $\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} a_{ij} a_{kl} E(V_i V_j V_k V_l) = \sum_{i=1}^{n} a_{ii}^2 E(V_i^4) + \left[\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ii}a_{jj} + \sum_{i=1}^{n}\sum_{j=1} a_{ij}a_{ij} + \sum_{i=1}^{n}\sum_{j=1} a_{ij}a_{ji}\right]$

$= \sum_{i=1}^{n} a_{ii}^2 E(V_i^4) + \text{Constant}$

Hence, we must minimize $\sum_{i=1}^{n} a_{ii}^2 E(V_i^4) \Rightarrow$ minimize $E(V_i^4)$

A useful inequality we have is $E(V_i^2)^2 \leq E(V_i^4)$ with constraint $E(V_i^2) = 1$

Since $E(V_i^2)^2 = 1^2 \leq E(V_i^4)$, $E(V_i^4)$ minimized if it equals 1.

Consider the distribution $\begin{cases} P(V_i = 1) = \frac{1}{2} \\ P(V_i = -1) = \frac{1}{2} \end{cases}$ This implies that $P(V_i^4 = 1) = 1$

Then, $E(V_i^4) = \sum_{m=1}^{1} x_m P(x_m) = 1$ (Since $x_m = 1$, $P(V_i^4 = x_m) = 1$)

Thus $Var(\hat{tr}(A))$ is minimized when $P(V_i = 1) = \frac{1}{2}$ and $P(V_i = -1) = \frac{1}{2}$ ∎

b) For $k=2$

Since $H = \begin{pmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{pmatrix}$

Then $H\begin{pmatrix} V \\ 0 \end{pmatrix} = \begin{pmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{pmatrix}\begin{pmatrix} V \\ 0 \end{pmatrix} = \begin{pmatrix} H_{11}V \\ H_{21}V \end{pmatrix}$

and $H\begin{pmatrix} H_{11}V \\ 0 \end{pmatrix} = \begin{pmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{pmatrix}\begin{pmatrix} H_{11}V \\ 0 \end{pmatrix} = \begin{pmatrix} H_{11}^2 V \\ H_{21}H_{11}V \end{pmatrix} = \begin{pmatrix} H_{11}^k V \\ H_{21}H_{11}^{k-1}V \end{pmatrix}$

Then consider it works for some $k \in \mathbb{N}$. We can see that it works for $k+1$

$$H\begin{pmatrix} H_{11}^{k-1}V \\ 0 \end{pmatrix} = \begin{pmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{pmatrix}\begin{pmatrix} H_{11}^{k-1}V \\ 0 \end{pmatrix} = \begin{pmatrix} H_{11}H_{11}^{k-1}V \\ H_{21}H_{11}^{k-1}V \end{pmatrix} = \begin{pmatrix} H_{11}^k V \\ H_{21}H_{11}^{k-1}V \end{pmatrix}$$

So this is true.

2. Suppose that $X_1, \cdots, X_n$ are independent Cauchy random variables with common density

$$f(x;\theta,\sigma) = \frac{1}{\pi\sigma}\left\{\frac{\sigma^2}{(x-\theta)^2+\sigma^2}\right\}$$

where $\theta \in (-\infty, \infty)$ and $\sigma > 0$ are unknown parameters. (θ is a location parameter and σ is a scale parameter.)

(a) Show that the median of this distribution is θ and the interquartile range is 2σ. Use this information to derive initial estimates for θ and σ in the Newton-Raphson algorithm (implemented in part (b)).

a) we want $\int_{-\infty}^{\theta} f(x;\theta,\sigma) = \frac{1}{2}$

Then $\int_{-\infty}^{\theta} \frac{1}{\pi\sigma}\left(\frac{\sigma^2}{(x-\theta)^2+\sigma^2}\right) dx$

$= \frac{1}{\pi\theta}\int_{-\infty}^{\theta} \frac{1}{\left(\frac{x-\theta}{\sigma}\right)^2+1} dx$

Let $u = \frac{x-\theta}{\sigma}$, then $du = \frac{1}{\sigma} dx$

Then, $\frac{1}{\pi}\int_{-\infty}^{\theta} \frac{1}{u^2+1} du = \frac{1}{\pi}\arctan(u)\Big|_{-\infty}^{\theta}$

$= \frac{1}{\pi}\arctan\left(\frac{x-\theta}{\sigma}\right)\Big|_{-\infty}^{\theta}$

$= 0 - \frac{1}{\pi}\left(-\frac{\pi}{2}\right)$

$= \frac{1}{2}$

Hence median of distribution is θ

$IQR = 2\sigma$

Find Q_3 first: $\frac{1}{\pi}\int_{-\infty}^{k} \frac{1}{u^2+1} du = 3/4$

$= \frac{1}{\pi}\arctan\left(\frac{x-\theta}{\sigma}\right)\Big|_{-\infty}^{k} = 3/4$

$\frac{1}{\pi}\arctan\left(\frac{k-\theta}{\sigma}\right) = \frac{3}{4} - \frac{1}{2}$

$\arctan\left(\frac{k-\theta}{\sigma}\right) = \frac{\pi}{4}$

$\frac{k-\theta}{\sigma} = \mathrm{Tan}\left(\frac{\pi}{4}\right) \Rightarrow k = \sigma + \theta$

Find Q_1 next: $\frac{1}{\pi}\int_{-\infty}^{m} \frac{1}{u^2+1} du = \frac{1}{4}$

$= \frac{1}{\pi}\arctan\left(\frac{x-\theta}{\sigma}\right)\Big|_{-\infty}^{m} = \frac{1}{4}$

$\frac{1}{\pi}\arctan\left(\frac{m-\theta}{\sigma}\right) = \frac{1}{4} - \frac{1}{2}$

$\arctan\left(\frac{m-\theta}{\sigma}\right) = -\frac{\pi}{4}$

$\frac{m-\theta}{\sigma} = \mathrm{Tan}\left(-\frac{\pi}{4}\right) \Rightarrow m = -\sigma + \theta$

Then IQR range is $k - m = \sigma + \theta - (-\sigma + \theta) = 2\sigma$ ∎

(b) Derive the likelihood equations for the MLEs of θ and σ and derive a Newton-Raphson algorithm for computing the MLEs based on $x_1, \cdots, x_n$. Implement this algorithm in R and test on data generated from a Cauchy distribution (using the R function `rcauchy`). Your function should also output an estimate of the variance-covariance matrix of the MLEs – this can be obtained from the Hessian of the maximized log-likelihood function.

MLE:

$$\ln(L(\theta,\sigma)) = \ln\left(\prod_{i=1}^{n} \frac{1}{\pi\sigma}\left(\frac{\sigma^2}{(x_i-\theta)^2+\sigma^2}\right)\right)$$

$$= \sum_{i=1}^{n} -\ln(\pi\sigma) + \ln(\sigma^2) - \ln((x_i-\theta)^2+\sigma^2)$$

$$= -n\ln(\pi\sigma) + n\ln(\sigma^2) - \sum_{i=1}^{n} \ln((x_i-\theta)^2+\sigma^2)$$

Score function: differentiate with respect to θ, σ

$$\frac{\partial \ln(L)}{\partial\theta} = \left[-n\ln(\pi\sigma) + n\ln(\sigma^2) - \sum_{i=1}^{n} \ln((x_i-\theta)^2+\sigma^2)\right]\frac{\partial}{\partial\theta}$$

$$= \sum_{i=1}^{n} \frac{2(x_i-\theta)}{(x_i-\theta)^2+\sigma^2}$$

$$\frac{\partial \ln(L)}{\partial\sigma} =$$

$$= -\frac{n}{\sigma} + \frac{2n}{\sigma} - \sum_{i=1}^{n} \frac{2\sigma}{(x_i-\theta)^2+\sigma^2}$$

$$S(\theta,\sigma) = \begin{bmatrix} \sum_{i=1}^{n} \frac{2(x_i-\theta)}{(x_i-\theta)^2+\sigma^2} \\ \frac{n}{\sigma} - \sum_{i=1}^{n} \frac{2\sigma}{(x_i-\theta)^2+\sigma^2} \end{bmatrix} \quad 2\times1 \text{ vector.}$$

Hessian:

$$\frac{\partial^2 \ln(L)}{\partial\theta^2} = \left[\sum_{i=1}^{n} \frac{2(x_i-\theta)}{(x_i-\theta)^2+\sigma^2}\right]\frac{\partial}{\partial\theta}$$

$$= \sum_{i=1}^{n} \frac{-2[(x_i-\theta)^2+\sigma^2] + 4(x_i-\theta)^2}{[(x_i-\theta)^2+\sigma^2]^2}$$

$$= \sum_{i=1}^{n} \frac{2(x_i-\theta)^2 - 2\sigma^2}{[(x_i-\theta)^2+\sigma^2]^2}$$

$$\frac{\partial \ln(L)}{\partial\theta\partial\sigma} = \left(\frac{n}{\sigma} - \sum_{i=1}^{n} \frac{2\sigma}{(x_i-\theta)^2+\sigma^2}\right)\frac{\partial}{\partial\theta}$$

$$= -\sum_{i=1}^{n} \frac{4\sigma(x_i-\theta)}{[(x_i-\theta)^2+\sigma^2]^2}$$

$$\frac{\partial^2 \ln(L)}{\partial\sigma^2} = -\frac{n}{\sigma^2} - \sum_{i=1}^{n} \frac{2[(x_i-\theta)^2+\sigma^2] - 4\sigma^2}{[(x_i-\theta)^2+\sigma^2]^2}$$

So, $$H(\theta,\sigma) = \begin{bmatrix} -\sum_{i=1}^{n} \frac{2(x_i-\theta)^2 - 2\sigma^2}{[(x_i-\theta)^2+\sigma^2]^2} & \sum_{i=1}^{n} \frac{4\sigma(x_i-\theta)}{[(x_i-\theta)^2+\sigma^2]^2} \\ \sum_{i=1}^{n} \frac{4\sigma(x_i-\theta)}{[(x_i-\theta)^2+\sigma^2]^2} & \frac{n}{\sigma^2} + \sum_{i=1}^{n} \frac{2[(x_i-\theta)^2+\sigma^2] - 4\sigma^2}{[(x_i-\theta)^2+\sigma^2]^2} \end{bmatrix}$$ 2x2 matrix

Notice the signs changed for second derivative for N-R Algorithm.

STA410 A3

Harold Hyun Woo Lee

11/23/2020

Question 1c)

```
#set.seed(1000)
leverage2 <- function(x, y, w, r=10, m=100) {
               #QR factorization
               qrx <- qr(x)
               qry <- qr(y)
               #Number of rows
               n <- nrow(x)
               #create leverage
               levx <- NULL
               levy <- NULL
               for (i in 1:m) {
                   v <- ifelse(runif(n)>0.5,1,-1)
                   v[-w] <- 0
                   v0 <- qr.fitted(qrx,v)
                   v1 <- qr.fitted(qry,v)
                   f <- v0
                   z <- v1
                   for (j in 2:r) {
                      v0[-w] <- 0
                      v1[-w] <- 0
                      v0 <- qr.fitted(qrx,v0)
                      v1 <- qr.fitted(qry,v1)
                      f <- f + v0/j
                      z <- z + v1/j
                      }
                   levx <- c(levx,sum(v*f))
                   levy <- c(levy,sum(v*z))

                   }
                std.err.x <- exp(-mean(levx))*sd(levx)/sqrt(m)
                levx <- 1 - exp(-mean(levx))
                std.err.y <- exp(-mean(levy))*sd(levy)/sqrt(m)
                levy <- 1 - exp(-mean(levy))
                r <- list(levx=levx,std.err.x=std.err.x,
                          levy=levy,std.err.y=std.err.y)
                return(r)
                }

x <- c(1:1000)/1000
X1 <- 1
```

```r
for (k in 1:5) X1 <- cbind(X1,cos(2*k*pi*x),sin(2*k*pi*x))
library(splines) # loads the library of functions to compute B-splines
X2 <- cbind(1,bs(x,df=10))

#plot(x,X2[,2])
#for (i in 3:11) points(x,X2[,i])


#create empty vector space
leverage_x <- c()
leverage_y <- c()
std.error_x <- c()
std.error_y <- c()

#run leverage function on every 50 rows of X1, X2
for (i in (1:20)){
  #Move the indices (w)
  help_lev <- leverage2(X1, X2, ((i*50-49):(50*i)), r = 10, m=100)
  #collect the leverages and standard errors of two models
  leverage_x <- c(leverage_x, help_lev$levx)
  leverage_y <- c(leverage_y, help_lev$levy)
  std.error_x <- c(std.error_x, help_lev$std.err.x)
  std.error_y <- c(std.error_y, help_lev$std.err.y)
}

soln = rbind(leverage_x, leverage_y, std.error_x, std.error_y)
soln
```

```
##                   [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
## leverage_x  0.54866350 0.51556872 0.60089093 0.48740169 0.53647855 0.55635121
## leverage_y  0.97005129 0.63276796 0.59273240 0.42530724 0.42702547 0.47915706
## std.error_x 0.05158312 0.03444945 0.05007482 0.04254350 0.04296375 0.04450109
## std.error_y 0.01545333 0.03770275 0.05004688 0.03967872 0.03761703 0.04352257
##                   [,7]       [,8]       [,9]      [,10]      [,11]      [,12]
## leverage_x  0.45721400 0.55417241 0.50056542 0.53675966 0.53278649 0.48995929
## leverage_y  0.29860317 0.49199306 0.33543749 0.42253763 0.41998342 0.32362879
## std.error_x 0.03840918 0.04373175 0.04870304 0.04619560 0.05203461 0.04590638
## std.error_y 0.02861253 0.04336732 0.03851681 0.04246154 0.04826690 0.03447928
##                  [,13]      [,14]      [,15]      [,16]      [,17]      [,18]
## leverage_x  0.49165921 0.50286223 0.51492930 0.56581334 0.42395359 0.46327660
## leverage_y  0.43224402 0.33177257 0.44694667 0.46107918 0.36071908 0.44592815
## std.error_x 0.04951322 0.04682095 0.04181707 0.04812813 0.03972241 0.03977997
## std.error_y 0.04779852 0.03584035 0.04025456 0.04448668 0.03684352 0.03925928
##                  [,19]      [,20]
## leverage_x  0.57434011 0.46594609
## leverage_y  0.68223157 0.93343701
## std.error_x 0.05305441 0.03886153
## std.error_y 0.05247248 0.02021234
```

We can see that g1 model has larger leverages than g2, model with B-spline functions except for the first 2 and last 2 leverages. We can also see that standard error of g1 and g2 are quite close to each other most of the time.

Question 2b)

```r
NewtonRaphson <- function(x, theta, sigma, iteration){
  n <- length(x)
  #Use median and IQR to derive initial estimates
  if (missing(theta)){
    theta = median(x)
    sigma = IQR(x)/2
  }
  alpha = c(theta, sigma)
  initial = alpha
  #Compute score function based on initial estimates
  score1 <- sum((2*(x-theta))/((x-theta)^2+sigma^2))
  score2 <- n/sigma - sum((2*sigma)/((x-theta)^2+sigma^2))

  score <- c(score1, score2)


  #compute Hessian Matrix
  H11 <- -sum((2*(x-theta)^2-(2*sigma^2))/(((x-theta)^2+sigma^2)^2))
  H12 <- sum((4*sigma*(x-theta))/(((x-theta)^2+sigma^2)^2))
  H21 <- sum((4*sigma*(x-theta))/(((x-theta)^2+sigma^2)^2))
  H22 <- n/(sigma^2) + sum((2*((x-theta)^2 + sigma^2) - 4*(sigma^2))/(((x-theta)^2
                                                                     +sigma^2)^2))

  H <- matrix(c(H11, H12, H21, H22), ncol=2, byrow = TRUE)

  #Newton-Raphson Iteration
  estimates <- c()
  for (i in (1:iteration)){
    alpha <- alpha + solve(H, score)

    #need to compute new scores
    score1_new <- sum((2*(x-alpha[1]))/((x-alpha[1])^2+alpha[2]^2))
    score2_new <- n/alpha[2] - sum((2*alpha[2])/((x-alpha[1])^2+alpha[2]^2))

    score_new <- c(score1_new, score2_new)

    #computing new Hessian
    H11_new <- -sum((2*(x-alpha[1])^2-(2*alpha[2]^2))/(((x-alpha[1])^2+alpha[2]^2)^2))
    H12_new <- sum((4*alpha[2]*(x-alpha[1]))/(((x-alpha[1])^2+alpha[2]^2)^2))
    H21_new <- sum((4*alpha[2]*(x-alpha[1]))/(((x-alpha[1])^2+alpha[2]^2)^2))
    H22_new <- n/(alpha[2]^2) + sum((2*((x-alpha[1])^2 + alpha[2]^2) -
                                   4*(alpha[2]^2))/(((x-alpha[1])^2+alpha[2]^2)^2))

    H_new <- matrix(c(H11_new, H12_new, H21_new, H22_new), ncol=2, byrow = TRUE)

    #overwrite new variables
    H <- H_new
    score <- score_new

    #putting our estimates in matrix form
    estimates <- rbind(estimates, alpha)
```

```r
  }
  #assign column and row names to estimates
  colnames(estimates) <- c("theta", "sigma")
  rownames(estimates) <- c(1:iteration)

  #Solving for variance-covariance matrix
  #Just the inverse of Hessian Matrix
  var_cov = solve(H)


  result = list(initial = initial, estimates = estimates, var_cov = var_cov)
  return(result)
}

set.seed(2)
x <- rcauchy(1000)
NewtonRaphson(x, iteration=10)
```

```
## $initial
## [1] 0.01022412 0.89195511
##
## $estimates
##          theta     sigma
## 1  0.009155243 0.9200896
## 2  0.009111783 0.9214914
## 3  0.009111696 0.9214947
## 4  0.009111696 0.9214947
## 5  0.009111696 0.9214947
## 6  0.009111696 0.9214947
## 7  0.009111696 0.9214947
## 8  0.009111696 0.9214947
## 9  0.009111696 0.9214947
## 10 0.009111696 0.9214947
##
## $var_cov
##              [,1]         [,2]
## [1,] 1.656910e-03 5.627663e-06
## [2,] 5.627663e-06 1.741860e-03
```

My example is rcauchy data of 1000 points and the algorithm used Newton-Raphson with 10 iterations. Notice that starting from the 3rd run, we have convergence. The function also outputs the variance covariance matrix (given by var_cov) of the MLE which is the inverse of the Hessian matrix.