MP HW3.3

Michael Pena

2024-02-27

```
#function to square root a matrix "A"
sqrtm <- function(A){
a <- eigen(A)
sqm <- a$vectors %*% diag(sqrt(a$values)) %*% t(a$vectors)
sqm <- (sqm+t(sqm))/2
}
#function for generating data
gen <- function(n,p,mu,sigma,seed){
#generate data from a p-variate normal with mean mu and covaraince sigma
#set seed to 2024
set.seed(seed)
#generate data from normal
z <- matrix(rnorm(n*p),n,p)
datan <- z %*% sqrtm(sigma) + matrix(mu,n,p,byrow = TRUE)
datan
}

# putting in the data
sig <- matrix(c(1,0.7,0.7,0.7,1,0.7,0.7,0.7,1), nrow = 3, ncol = 3)
mu <- matrix(c(-1,1,2), nrow =3)
x <- gen(200,3,mu,sig,2025)
```

```
# make gradient
gradient <- function(x,mu,sig){
  p <- nrow(sig)
  n <- nrow(x)
  inv.sig <- solve(sig)
  # set initials
  xi.sum <- matrix(0, p, 1)
  C.mu <- matrix(0, p, p)
  # compute sum of Xi and sum C(mu)
  for(i in 1:n){
    xi <- x[i,] - mu
    xi.sum <- xi.sum + xi
    C.mu <- C.mu + xi %*% t(xi)
  }
  # place elements into gradient mu and gradient sig
  grad.mu <- inv.sig %*% xi.sum
  A <- (n * inv.sig) - inv.sig %*% C.mu %*% inv.sig
  grad.sig <- matrix(0, nrow = nrow(A), ncol = ncol(A))
  #gradient sig
  for(i in 1:nrow(sig)){
    grad.sig[i,i] <- -(1/2) * A[i,i]
```

```
  }
  for(i in 1:nrow(sig)-1){
    for (j in (i+1):ncol(sig)){
      grad.sig[i,j] <- -1 * A[i,j]
      grad.sig[j,i] <- grad.sig[i,j]
    }
  }
  grad.norm <- norm(to.theta(grad.mu,grad.sig), type = '2')
  list(grad.mu = grad.mu, grad.sig = grad.sig, grad.norm = grad.norm)
 }
```

```
# Hessian Matrix
hessian <- function(x, mu, sigma) {
  n <- nrow(x)
  p <- ncol(x)
  siginv <- solve(sigma)
  mu_hess <- -n * siginv # second derivative of dmu,dmu

  # initialize matrix for hessian of dsig,dsig
  sig_hess <- matrix(0, nrow = p * (p + 1) / 2, ncol = p * (p + 1) / 2)

  # initialize C matrix, calculate C(mu)
  C <- matrix(0, nrow = p, ncol = p)
  sxm <- matrix(0, p, 1)  # initialize sxm
  for(i in 1:n) {
    xm <- x[i,] - mu  # compute each xi - mu
    sxm <- sxm + xm  # sum of xi - mu
    # now to find C = sum(xi-mu)(xi-mu)^(T)
    C <- C + xm %*% t(xm)
  }
  Z <- (-1/2)*((-n*diag(p)+2* siginv %*% C) %*% siginv)

  # calculating the hessian matrix
  c_count <- 0
  r_count <- 0
  for(i in 1:p) {
    for(j in 1:i) {
      r_count <- r_count + 1
      c_count <- 0
      for(k in 1:p) {
        for(l in 1:k) {
          c_count <- c_count + 1
          if(i == j && k == l) {
            sig_hess[r_count, c_count] <- Z[k,i] * siginv[i,k]
          } else if(i != j && k != l) {
            sig_hess[r_count, c_count] <- Z[k,i] * siginv[j,l] + Z[l,j] * siginv[i,k] + Z[k,j] * siginv
          } else if(i != j && k == l) {
            sig_hess[r_count, c_count] <- Z[k,i] * siginv[j,k] + Z[k,j] * siginv[i,k]
          } else if(i == j && k != l) {
            sig_hess[r_count, c_count] <- Z[l,i] * siginv[i,k] + Z[k,i] * siginv[i,l]
          }
        }
      }
    }
```

```
  }

  # dmu,dsigma
  mu_sig_hess <- matrix(0, nrow = p, ncol = p * (p + 1) / 2)
  sxm2 <- matrix(0,p,1)
  for(i in 1:n){
    xm = x[i,] - mu
    sxm2 = sxm2 + xm
  }
  D <- -siginv %*% sxm2
  r_count <- 0
  c_count <- 0
  for(i in 1:p){
    r_count = r_count +1
    c_count = 0
  for(k in 1:p) {
    for(l in 1:k) {
      c_count <- c_count + 1
      if(k == l) {
        mu_sig_hess[r_count, c_count] <- D[l,] * siginv[i,k]
      } else if(k != l) {
        mu_sig_hess[r_count, c_count] <- D[l,] * siginv[k,i] + D[k,] * siginv[l,i]
      }
    }
  }
  }
  hessian <- cbind(rbind(mu_hess, t(mu_sig_hess)), rbind(mu_sig_hess, sig_hess))
  hessian <- 0.5 * (hessian + t(hessian))
  return(hessian)
}
 # turn theta into a mu and sigma
 from.theta <- function(p,theta){
   mu <- theta[1:p]
   sig <- matrix(0, nrow = p, ncol = p)

   k = p + 1

   for (i in 1:p){
     for (j in 1:i){
       sig[i,j] <- theta[k]
       sig[j,i] <- sig[i,j]
       k = k + 1
     }
   }
  list(mu = mu, sig = sig)
 }

# # compile Sigma and Mu into a single theta vector
 to.theta <- function(mu,sig){
   p <- nrow(sig)
   theta <- matrix(0,nrow = p + p*(1+p)/2,ncol = 1)
   theta[1:p] <- mu
```

```r
  k = p + 1
  for(i in 1:p){
    for(j in 1:i){
      theta[k] <- sig[i,j]
      k = k + 1
    }
  }
  return(theta)
 }
```

```r
#likelihood function
likemvn <- function (x,mu,sig) {
  # computes the likelihood and the gradient for multivariate normal
  n = nrow(x)
  p = ncol(x)

  sig.inv <- solve(sig)
  C.mu = matrix(0,p,p) # initializing sum of (xi-mu)(xi-mu)^T
  xi.sum = matrix(0,p,1) # initializing sum of xi-mu
  for (i in 1:n){
    xi = x[i,] - mu
    C.mu = C.mu + xi %*% t(xi)
  }

  ell = -(n*p*log(2*pi)+n*log(det(sig)) + sum(sig.inv * C.mu ))/2
  return(ell)
}
```

```r
#Newton Method Function
newton <- function(x, mu, sig, maxit, tolerr, tolgrad){
  header = paste0("Iteration", "      halving", "     log-likelihood","      ||Gradient||")
  print(header)

  for (it in 1:maxit) {
    # first steps
    theta0 <- to.theta(mu, sig)
    L0 <- likemvn(x, mu, sig)
    # gradient elements
    grad.on.mu <- gradient(x, mu, sig)$grad.mu
    grad.on.sig <- gradient(x,mu, sig)$grad.sig
    grad.on.norm <- gradient(x,mu,sig)$grad.norm

    #calculate direction vector
    hess <- hessian(x,mu,sig)
    inv.hess <- solve(hess)
    direc <- (-1)*(inv.hess %*% to.theta(grad.on.mu, grad.on.sig))
    # print

    print(sprintf('%2.0f                 --          %3.4f                %.1e',
                 it,  L0, grad.on.norm))



    # get new parameters
```

```
    theta1 <- theta0 + direc
    mu1 <- from.theta(3, theta1)$mu
    sig1 <- from.theta(3, theta1)$sig
    grad.on.norm1 <- gradient(x, mu1, sig1)$grad.norm

    # print NA if e-vals are not positive
    if(all(eigen(sig1)$values >0)){L1 <- likemvn(x,mu1,sig1)}
    else{L1 <- NaN}

    halve <- 0
    print(sprintf('%2.0f                 %2.0f          %3.4f                %.1e',
                  it,  halve, L1, grad.on.norm1))

    # step-halving T_T
    while((all(eigen(sig1)$values >0) == FALSE & halve < 20) || L1 < L0){
      halve = halve + 1
      theta1 <- theta0 + direc/(2^halve)
      mu1 <- from.theta(3, theta1)$mu
      sig1 <- from.theta(3, theta1)$sig
      if(all(eigen(sig1)$values >0)){L1 <- likemvn(x,mu1,sig1)}
      else{L1 <- NaN}

      # get the new norm grad
      grad.on.norm1 <- gradient(x, mu1, sig1)$grad.norm

      print(sprintf('%2.0f                 %2.0f          %3.4f                %.1e',
                    it,  halve, L1, grad.on.norm1))
    }
      print("-----------------------------------------------------------------")
      print(header)

    theta0 <- theta1

    # stopping conditions
    r.e = max(abs(theta0 - theta1)/abs(pmax(1,abs(theta0))))
    if (r.e < tolerr & grad.on.norm1 < tolgrad){break}
    mu <- mu1
    sig <- sig1
  }
   return(list("estimator of mu"=mu, "estimator of sigma" = sig, "iteration" = it))

}
```

Part (a).

```
mu.0 <- matrix(c(-1.5,1.5,2.3),ncol = 1)
sig.0 <- matrix(c(1,0.5,0.5,
                  0.5,1,0.5,
                  0.5,0.5,1), nrow=3, ncol=3)
newton(x,mu.0,sig.0,500,1e-07,1e-07)
```

```
## [1] "Iteration      halving     log-likelihood      ||Gradient||"
## [1] " 1                --          -838.6352               3.9e+02"
```

```
## [1] " 1                 0          NaN               7.0e+02"
## [1] " 1                 1          NaN               1.1e+03"
## [1] " 1                 2          NaN               2.6e+05"
## [1] " 1                 3          NaN               9.5e+03"
## [1] " 1                 4          -776.8361               3.6e+02"
## [1] "-----------------------------------------------------------------"
## [1] "Iteration      halving     log-likelihood      ||Gradient||"
## [1] " 2                --          -776.8361               3.6e+02"
## [1] " 2                 0          -10769.1064               2.2e+06"
## [1] " 2                 1          -722.4349               5.0e+02"
## [1] "-----------------------------------------------------------------"
## [1] "Iteration      halving     log-likelihood      ||Gradient||"
## [1] " 3                --          -722.4349               5.0e+02"
## [1] " 3                 0          -704.8749               1.8e+02"
## [1] "-----------------------------------------------------------------"
## [1] "Iteration      halving     log-likelihood      ||Gradient||"
## [1] " 4                --          -704.8749               1.8e+02"
## [1] " 4                 0          -699.9388               5.3e+01"
## [1] "-----------------------------------------------------------------"
## [1] "Iteration      halving     log-likelihood      ||Gradient||"
## [1] " 5                --          -699.9388               5.3e+01"
## [1] " 5                 0          -699.1587               9.1e+00"
## [1] "-----------------------------------------------------------------"
## [1] "Iteration      halving     log-likelihood      ||Gradient||"
## [1] " 6                --          -699.1587               9.1e+00"
## [1] " 6                 0          -699.1275               4.2e-01"
## [1] "-----------------------------------------------------------------"
## [1] "Iteration      halving     log-likelihood      ||Gradient||"
## [1] " 7                --          -699.1275               4.2e-01"
## [1] " 7                 0          -699.1274               9.8e-04"
## [1] "-----------------------------------------------------------------"
## [1] "Iteration      halving     log-likelihood      ||Gradient||"
## [1] " 8                --          -699.1274               9.8e-04"
## [1] " 8                 0          -699.1274               5.5e-09"
## [1] "-----------------------------------------------------------------"
## [1] "Iteration      halving     log-likelihood      ||Gradient||"

## $`estimator of mu`
## [1] -0.9915895  0.9938698  2.0319713
##
## $`estimator of sigma`
##           [,1]      [,2]      [,3]
## [1,] 0.9176861 0.6112407 0.6902985
## [2,] 0.6112407 0.9727364 0.7691457
## [3,] 0.6902985 0.7691457 1.1088343
##
## $iteration
## [1] 8
```

Part (b).

```
# building the fisher matrix
fisher <- function(x,sig){
```

```r
  U <- solve(sig)
  p <- ncol(x)
  p1 <- p*(1 +p)/2
  n <- nrow(x)

  # produce the dmu-dmu partition
  dmdm <- matrix(0,p,p)
  for(i in 1:p){
    for(j in 1:i){
      dmdm[i,j] <- -n*U[i,j]
      dmdm[j,i] <- -n*U[i,j]
    }
  }

  # produce the dsig-dsig partition
  dsds <- matrix(0,p1,p1)

  v <- matrix(c(1,1,
                2,1,
                2,2,
                3,1,
                3,2,
                3,3), nrow = 6, ncol = 2, byrow = TRUE)
  for(v1 in 1:6){
    for(v2 in 1:6){
      i = as.numeric(v[v1,1])
      j = as.numeric(v[v1,2])
      k = as.numeric(v[v2,1])
      l = as.numeric(v[v2,2])
      if(i == j && k == l){ #case 1
        dsds[v1,v2] <- U[k,i] * U[i,k]
      } else if(i != j && k != l){ # case 2
        dsds[v1,v2] <- U[k,i] * U[j,l] + U[l,j] * U[i,k] + U[k,j] * U[i,l] + U[l,i] * U[j,k]
      } else if(i != j && k == l){ # case 3
        dsds[v1,v2] <- U[k,i] * U[j,k] + U[k,j] * U[i,k]
      } else { #case 4
        dsds[v1,v2] <- U[l,i] * U[i,k] + U[k,i] * U[i,l]
      }
    }
  }
  dsds <- (-n/2)*dsds

  # bind matrices
  o1 <- matrix(0,p,p1)
  o2 <- t(o1)
  F <- rbind(cbind(dmdm,o1),cbind(o2,dsds))
  F <- (-1)*F
  return(F)
}

fisher(x,sig.0)
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
##  [1,]  300 -100 -100    0    0    0    0    0    0
```

```
##  [2,] -100  300 -100    0    0    0    0    0    0
##  [3,] -100 -100  300    0    0    0    0    0    0
##  [4,]    0    0    0  225 -150   25 -150   50   25
##  [5,]    0    0    0 -150  500 -150 -100 -100   50
##  [6,]    0    0    0   25 -150  225   50 -150   25
##  [7,]    0    0    0 -150 -100   50  500 -100 -150
##  [8,]    0    0    0   50 -100 -150 -100  500 -150
##  [9,]    0    0    0   25   50   25 -150 -150  225
```

```
# the algorithm for fisher stepping
newton_fisher <- function(x, mu, sig, maxit, tolerr, tolgrad){
  header = paste0("Iteration", "      halving", "     log-likelihood","      ||Gradient||")
  print(header)

  for (it in 1:maxit) {
    # first steps
    theta0 <- to.theta(mu, sig)
    L0 <- likemvn(x, mu, sig)
    # gradient elements
    grad.on.mu <- gradient(x, mu, sig)$grad.mu
    grad.on.sig <- gradient(x,mu, sig)$grad.sig
    grad.on.norm <- gradient(x,mu,sig)$grad.norm

    #calculate direction vector
    fish <- fisher(x,sig)
    inv.fish <- solve(fish)
    direc <- (inv.fish %*% to.theta(grad.on.mu, grad.on.sig))
    # print

    print(sprintf('%2.0f                --          %3.4f                %.1e',
                it,  L0, grad.on.norm))



    # get new parameters
    theta1 <- theta0 + direc
    mu1 <- from.theta(3, theta1)$mu
    sig1 <- from.theta(3, theta1)$sig
    grad.on.norm1 <- gradient(x, mu1, sig1)$grad.norm

    # print NA if e-vals are not positive
    if(all(eigen(sig1)$values >0)){L1 <- likemvn(x,mu1,sig1)}
    else{L1 <- NaN}

    halve <- 0
    print(sprintf('%2.0f                %2.0f          %3.4f                %.1e',
                it,  halve, L1, grad.on.norm1))

    # step-halving T_T
    while((all(eigen(sig1)$values >0) == FALSE & halve < 20) || L1 < L0){
      halve = halve + 1
      theta1 <- theta0 + direc/(2^halve)
      mu1 <- from.theta(3, theta1)$mu
      sig1 <- from.theta(3, theta1)$sig
```

```r
      if(all(eigen(sig1)$values >0)){L1 <- likemvn(x,mu1,sig1)}
      else{L1 <- NaN}

      # get the new norm grad
      grad.on.norm1 <- gradient(x, mu1, sig1)$grad.norm

      print(sprintf('%2.0f                  %2.0f          %3.4f                %.1e',
                    it,  halve, L1, grad.on.norm1))
    }
      print("------------------------------------------------------------------")
      print(header)

    theta0 <- theta1

    # stopping conditions
    r.e = max(abs(theta0 - theta1)/abs(pmax(1,abs(theta0))))
    if (r.e < tolerr & grad.on.norm1 < tolgrad){break}
    mu <- mu1
    sig <- sig1
  }
   return(list("estimator of mu"=mu, "estimator of sigma" = sig, "iteration" = it))

}

newton_fisher(x,mu.0,sig.0,500,1e-07,1e-07)
```

```
## [1] "Iteration      halving     log-likelihood      ||Gradient||"
## [1] " 1                --          -838.6352              3.9e+02"
## [1] " 1                 0          -733.7971              8.4e+01"
## [1] "------------------------------------------------------------------"
## [1] "Iteration      halving     log-likelihood      ||Gradient||"
## [1] " 2                --          -733.7971              8.4e+01"
## [1] " 2                 0          -699.1274              4.0e-13"
## [1] "------------------------------------------------------------------"
## [1] "Iteration      halving     log-likelihood      ||Gradient||"
```

```
## $`estimator of mu`
## [1] -0.9915895  0.9938698  2.0319713
##
## $`estimator of sigma`
##           [,1]      [,2]      [,3]
## [1,] 1.1761677 0.3539183 0.5540296
## [2,] 0.3539183 1.2289047 0.9048035
## [3,] 0.5540296 0.9048035 1.1806739
##
## $iteration
## [1] 2
```