Math 534 Homework 3.3

Newton and Fisher-Scoring Methods
Mike Palmer
due 2024/02/28

Exercise J-2.2 In this exercise, we assume we have a set of data $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ from a p-variate normal distribution with mean $\boldsymbol{\mu} = [\mu_1, \mu_2, \ldots, \mu_p]^T$ and a $p \times p$ covariance matrix $\boldsymbol{\Sigma} = (\sigma_{ij})$. Write a general function to maximize the following log-likelihood function with respect to parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$:

$$\ell(\boldsymbol{\mu}, \boldsymbol{\Sigma}|\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n) = -\frac{1}{2}\left\{ nplog(2\pi) + nlog(|\boldsymbol{\Sigma}|) + trace\left[\boldsymbol{\Sigma}^{-1}c(\boldsymbol{\mu})\right] \right\},$$

$$\text{where } c(\boldsymbol{\mu}) = \sum_{z=1}^{n} (\boldsymbol{x}_z - \boldsymbol{\mu})(\boldsymbol{x}_z - \boldsymbol{\mu})^T.$$

There are p parameters in $\boldsymbol{\mu}$ and $p(p+1)/2$ parameters in $\boldsymbol{\Sigma}$ (since $\sigma_{ij} = \sigma_{ji}$). Define

$$\boldsymbol{\theta} = [\mu_1, \mu_2, \ldots, \mu_p, \sigma_{11}, \sigma_{21}, \sigma_{22}, \sigma_{31}, \sigma_{32}, \sigma_{33}, \ldots, \sigma_{p1}, \sigma_{p2}, \ldots, \sigma_{pp}]^T.$$

Write a general code that applies each of the following methods to obtain the maximum likelihood estimate of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ for a given set of $n \times p$ matrix of data:

II. Newton's method with step-halving
III. Fisher-Scoring algorithm with step halving

```
library(knitr) #style output
library(kableExtra) #style output
library(dplyr) #style output
```

Data Generation Generate 200 data points from a trivariate normal distribution with the following parameters for $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$.

$$\boldsymbol{\mu} = [-1, 1, 2]^T \text{ and } \boldsymbol{\Sigma} = \begin{pmatrix} 1 & 0.7 & 0.7 \\ 0.7 & 1 & 0.7 \\ 0.7 & 0.7 & 1 \end{pmatrix}$$

```r
# Generate data
sqrtm <- function (A) {
  # Obtain matrix square root of a matrix A
  a = eigen(A)
  sqm = a$vectors %*% diag(sqrt(a$values)) %*% t(a$vectors)
  sqm = (sqm+t(sqm))/2
}

gen <- function(n,p,mu,sig,seed = 534){
  #---- Generate data from a p-variate normal with mean mu and covariance sigma
  # mu should be a p by 1 vector
  # sigma should be a positive definite p by p matrix
  # Seed can be optionally set for the random number generator
  set.seed(seed)
  # generate data from normal mu sigma
  z = matrix(rnorm(n*p),n,p)
  datan = z %*% sqrtm(sig) + matrix(mu,n,p, byrow = TRUE)
  datan
}

mu = matrix(c(-1,1,2),nrow = 3,ncol = 1)
sigma = matrix(c(1,.7,.7,.7,1,.7,.7,.7,1),nrow = 3,ncol = 3)
data = gen(200,3,mu,sigma,seed = 2025)
data[1:3,]
```

```
##            [,1]       [,2]      [,3]
## [1,] -0.5042864  1.0483093 2.1785941
## [2,] -2.1913297 -1.7714460 0.3435119
## [3,] -0.8181978  0.3721832 1.3244742
```

```r
#loglikelihood as a separate R function
loglike_f <- function(data,mu,sigma){
  n = nrow(data)
  p = ncol(data)
  c_mu = matrix(0,nrow = p, ncol = p) #pxp #c_mu like c(\mu) from previous hw
  for(i in 1:n){ c_mu = c_mu + (data[i,] - mu) %*% t(data[i,] - mu) }
  l = -1/2*(n*p*log(2*pi)+n*log(det(sigma))+sum(diag(solve(sigma) %*% c_mu))) #how this note? Note that
  list(l=l)
}

#grad check
#mu_hat = colMeans(data) #sig_hat = (nrow(data)-1)*cov(data)/nrow(data)
#grad_mu_loglike_f(data,mu_hat,sig_hat) =0 #grad_sigma_loglike_f(data,mu_hat,sig_hat) should = 0
#wrt mu #gradient of loglikelihood as a separate R function
grad_mu_loglike_f <- function(data,mu,sigma){
  n = nrow(data)
  p = ncol(data)
  d_c_mu = matrix(0,nrow = p, ncol = 1) #px1 #d_c_mu as in differential of c_mu #same as sxm
  for(i in 1:n){ d_c_mu = d_c_mu + (data[i,] - mu) }
  grad_mu = solve(sigma) %*% d_c_mu
  grad_mu
}

#wrt sigma #gradient of loglikelihood as a separate R function
grad_sigma_loglike_f <- function(data,mu,sigma){
  n = nrow(data)
  p = ncol(data)
  c_mu = matrix(0,nrow = p, ncol = p) #pxp
  for(i in 1:n){ c_mu = c_mu + (data[i,] - mu) %*% t(data[i,] - mu) }
  grad_sigma = -n/2 * solve(sigma) %*% (sigma - c_mu/n) %*% solve(sigma)
  grad_sigma
}

#input mu and sigma, output teta vector
mu_sigma_to_teta_vec <- function(mu,sigma, is.gradient = FALSE){

  p = nrow(mu)
  teta = matrix(0,nrow =p+p*(p+1)/2, ncol = 1)
  teta[1:p,] = mu
  for (i in 1:p){    #teta[(p+1) to p(p+1)/2,] = sigma
    for (j in 1:i){
      p = p+1
      if(is.gradient == FALSE){
        teta[p,] = sigma[i,j]
      }
      else{
        if(i == j){
          teta[p,] = sigma[i,j]
        }
        else {
          teta[p,] = 2*sigma[i,j]
        }
      }
```

```r
    }
  }

  if(is.gradient == FALSE) return(list(teta = teta, mu =  mu, sigma = sigma))
  if(is.gradient == TRUE)  return(list(grad_teta = teta, grad_mu =  mu, grad_sigma = sigma))
}

#input teta vector, output mu and sigma
teta_vec_to_mu_sigma <- function(teta_vec,p){

  mu =  matrix(teta_vec[1:p],nrow = p, ncol = 1)
  sigma =  matrix(0,nrow = p, ncol = p) #sigma = teta_vec[(p+1) to p(p+1)/2,]
  for (i in 1:p) {
    for (j in 1:i) {
      p = p+1
      sigma[i,j] = teta_vec[p]
      if(i != j) sigma[j,i] = teta_vec[p]
    }
  }

  list(mu = mu, sigma = sigma)
}

###### hessian

#H = matrix(0,nrow =p+p*(p+1)/2, ncol = p+p*(p+1)/2)
H <- function(data,mu,sigma){

  n = nrow(data)
  p = length(mu)
  sigma_inv = solve(sigma)
  c_mu = matrix(0,nrow = p, ncol = p) #pxp
  for(i in 1:n){ c_mu = c_mu + (data[i,] - mu) %*% t(data[i,] - mu) }
  s_x_mu = matrix(0,nrow = p, ncol = 1) #pxp
  for(i in 1:n){ s_x_mu = s_x_mu + (data[i,] - mu) }

  #dmudmu
  H_mumu = matrix(0,nrow =p, ncol = p)
  rcnt = 0 #p
  for (i in 1:p) {
    rcnt = rcnt + 1
    ccnt = 0 #p
    for (j in 1:p) {
      ccnt = ccnt+1
      H_mumu[rcnt,ccnt] = -n*sigma_inv[i,j]
    }
  }

  #H_mumu

  #dmudsigma
  A = sigma_inv %*% s_x_mu
  H_musigma = matrix(0,nrow = p*(p+1)/2, ncol = p)
```

```
ccnt = 0
for (k in 1:p){
  rcnt = 0
  ccnt = ccnt + 1
  for (i in 1:p) {

    for (j in 1:i) {

      rcnt = rcnt + 1

      if (i == j){
        H_musigma[rcnt,ccnt] = -sigma_inv[k,i] * A[i]
      }

      else if (i != j){
        H_musigma[rcnt,ccnt] = -(sigma_inv[k,i] * A[j] + sigma_inv[k,j] * A[i])
      }

    }
  }
}

#H_musigma

#dsigmadsigma
A = n*(sigma_inv/2 - (sigma_inv %*% (c_mu/n) %*% sigma_inv))
#A = n*((diag(p)/2 - sigma_inv %*% (c_mu/n)) %*% sigma_inv)
#A = (-1/2)*((-n*diag(p) + 2 * sigma_inv %*% c_mu) %*% sigma_inv)
H_sigmasigma = matrix(0,nrow =p*(p+1)/2, ncol = p*(p+1)/2)

rcnt = 0
for (i in 1:p) {

  for (j in 1:i) {

    rcnt = rcnt + 1
    ccnt = 0
    for (k in 1:p) {

      for (l in 1:k) {
        ccnt = ccnt + 1

        if (i==j & k==l){

          H_sigmasigma[rcnt,ccnt] = A[i,k]*sigma_inv[l,j]
          #H_sigmasigma[rcnt,ccnt] = A[k,i]*sigma_inv[j,l]
          #H_sigmasigma[rcnt,ccnt] = A[i,k]*sigma_inv[l,j]
          #H_sigmasigma[rcnt,ccnt] = 1000*i+100*j+10*k+l

        }

        else if (i!=j & k==l){
```

```
            H_sigmasigma[rcnt,ccnt] = A[i,k]*sigma_inv[l,j] + A[j,k]*sigma_inv[l,i]
            #H_sigmasigma[rcnt,ccnt] = A[k,i]*sigma_inv[j,l] + A[k,j]*sigma_inv[i,l]
            #H_sigmasigma[rcnt,ccnt] = A[i,k]*sigma_inv[l,j] + A[j,k]*sigma_inv[l,i]
            #H_sigmasigma[rcnt,ccnt] = 1000*i+100*j+10*k+l

          }

          else if (i==j & k!=l){

            H_sigmasigma[rcnt,ccnt] = A[i,l]*sigma_inv[k,j] + A[i,k]*sigma_inv[l,j]
            #H_sigmasigma[rcnt,ccnt] = A[l,i]*sigma_inv[j,k] + A[k,i]*sigma_inv[i,l]
            #H_sigmasigma[rcnt,ccnt] = A[i,l]*sigma_inv[k,j] + A[i,k]*sigma_inv[l,j]
            #H_sigmasigma[rcnt,ccnt] =  1000*i+100*j+10*k+l

          }

          else if (i!=j & k!=l){

            H_sigmasigma[rcnt,ccnt] = A[i,l]*sigma_inv[k,j] + A[i,k]*sigma_inv[l,j] + A[j,l]*sigma_inv[l
            #H_sigmasigma[rcnt,ccnt] = A[k,i]*sigma_inv[j,l] + A[l,j]*sigma_inv[i,k] + A[k,j]*sigma_inv
            #H_sigmasigma[rcnt,ccnt] = A[i,l]*sigma_inv[k,j] + A[i,k]*sigma_inv[l,j] + A[j,l]*sigma_inv
            #H_sigmasigma[rcnt,ccnt] = 1000*i+100*j+10*k+l
          }

        }
      }

    }
  }

  #H_mumu
  #H_musigma
  #H_sigmasigma

  H = rbind(cbind(H_mumu,t(H_musigma)),
            cbind(H_musigma,H_sigmasigma))
  return(H)
}

#newton
#direction = -inverse(hessian)*gradient
#note hessian is jacobian of gradient
```

```
newton <- function(data, mu_start = NULL, sigma_start = NULL,
                              maxit = 500, tolerr = 1e-6, tolgrad = 1e-5,
                              #teta_star = NULL, #convergence_power = (1+sqrt(5))/2,
                              show = NULL){

  it = 1; stop = FALSE; for_show = matrix(0,nrow = 0,ncol = 4); p = length(mu_start)
  teta_n = mu_sigma_to_teta_vec(mu_start,sigma_start, is.gradient = FALSE)$teta #starting point


  while(it <= maxit & stop == FALSE){   #core calculation
```

```
mu_n = teta_vec_to_mu_sigma(teta_n,p=p)$mu
sigma_n = teta_vec_to_mu_sigma(teta_n,p=p)$sigma
f_teta_n = loglike_f(data,mu_n,sigma_n)$l #check for positive definite???? or throw error at beginn
grad_mu_n = grad_mu_loglike_f(data,mu_n,sigma_n)
grad_sigma_n = grad_sigma_loglike_f(data,mu_n,sigma_n)
grad_teta_n = mu_sigma_to_teta_vec(grad_mu_n,grad_sigma_n, is.gradient = TRUE)$grad_teta
hess_inv = solve(H(data,mu_n,sigma_n))


teta_n_new = teta_n + (-hess_inv %*% grad_teta_n) # Steepest Ascent #dir =  -Hess_inv* grad_teta_n

#need sigma to be positive definite aka positive eigenvalues    or
#pos_definite = all(diag(teta_vec_to_mu_sigma(teta_n_new,p=3)$sigma)>0)
pos_definite = all(eigen(teta_vec_to_mu_sigma(teta_n_new,p=p)$sigma)$values>0)

if(pos_definite){
  mu_n_new = teta_vec_to_mu_sigma(teta_n_new,p=p)$mu
  sigma_n_new = teta_vec_to_mu_sigma(teta_n_new,p=p)$sigma
  f_teta_n_new = loglike_f(data,mu_n_new,sigma_n_new)$l
  grad_mu_n_new = grad_mu_loglike_f(data,mu_n_new,sigma_n_new) #needed if not halving
  grad_sigma_n_new = grad_sigma_loglike_f(data,mu_n_new,sigma_n_new) #needed if not halving
  grad_teta_n_new = mu_sigma_to_teta_vec(grad_mu_n_new,grad_sigma_n_new, is.gradient = TRUE)$grad_t
}

for_show = rbind(for_show,c(it, NaN, f_teta_n, norm(grad_teta_n, type = "2")))
halve = 0
while ( halve <= 20 &  (pos_definite == FALSE || f_teta_n_new < f_teta_n )){

  teta_n_new = teta_n + (-hess_inv %*% grad_teta_n)/2^halve  # Steepest Ascent #dir = grad_teta_n #

  #need sigma to be positive definite aka positive eigenvalues    or
  #pos_definite = all(diag(teta_vec_to_mu_sigma(teta_n_new,p=3)$sigma)>0)
  pos_definite = all(eigen(teta_vec_to_mu_sigma(teta_n_new,p=p)$sigma)$values>0)

  if(pos_definite){
    mu_n_new = teta_vec_to_mu_sigma(teta_n_new,p=p)$mu
    sigma_n_new = teta_vec_to_mu_sigma(teta_n_new,p=p)$sigma

    #f_teta_n = loglike_f(data,mu_n,sigma_n)$l
    f_teta_n_new = loglike_f(data,mu_n_new,sigma_n_new)$l

    grad_mu_n_new = grad_mu_loglike_f(data,mu_n_new,sigma_n_new)
    grad_sigma_n_new = grad_sigma_loglike_f(data,mu_n_new,sigma_n_new)
    grad_teta_n_new = mu_sigma_to_teta_vec(grad_mu_n_new,grad_sigma_n_new, is.gradient = TRUE)$grad_

    L2_norm = norm(grad_teta_n_new, type = "2")
    for_show = rbind(for_show,c(it, halve, f_teta_n_new, L2_norm))
  }
  else{
    for_show = rbind(for_show,c(it, halve, NaN, NaN))
  }

  halve = halve + 1
```

```
    }

    #stop calculation #aka convergence?    #write function to check for convergence?
    mod_rel_err = max(abs(teta_n_new-teta_n)/pmax(1,abs(teta_n_new)))
    L2_norm = norm(grad_teta_n_new, type = "2") #needed if not halving
    if (mod_rel_err<tolerr & L2_norm < tolgrad) stop = TRUE

    teta_n <- teta_n_new #next iteration
    it = it + 1
    }



  #print estimates
  mu_print = data.frame(`mu`=teta_vec_to_mu_sigma(teta_n_new,p=p)$mu)
  sigma_print = data.frame(`sigma` = teta_vec_to_mu_sigma(teta_n_new,p=p)$sigma)
  colnames(sigma_print) = c('sigma',rep('', p-1))

  print(kable(list(mu_print,sigma_print),
        align = 'c',
        booktabs = TRUE,
        caption = "Estimates"
        )
        %>% kable_styling(latex_options = "HOLD_position")
        )

  #print iterations
  if(show == "show_2"){
    for_show = for_show[for_show[,1]==1 | for_show[,1]==2 | for_show[,1]== (it-2) | for_show[,1]== (it-

  desc = data.frame(`it`=for_show[,1],`halve`=for_show[,2],`loglikelihood`=for_show[,3],`L2_norm`=for_s

  return(kable(desc, col.names = names(desc), align = "cccc", booktabs = TRUE, caption = 'Iterations') 

}
```

```r
###### Information Matrix

#I = matrix(0,nrow =p+p*(p+1)/2, ncol = p+p*(p+1)/2)

I <- function(data,mu,sigma){

  n = nrow(data)
  p = length(mu)
  sigma_inv = solve(sigma)
  c_mu = matrix(0,nrow = p, ncol = p) #pxp
  for(i in 1:n){ c_mu = c_mu + (data[i,] - mu) %*% t(data[i,] - mu) }
  s_x_mu = matrix(0,nrow = p, ncol = 1) #pxp
  for(i in 1:n){ s_x_mu = s_x_mu + (data[i,] - mu) }

  #dmudmu
  H_mumu = matrix(0,nrow =p, ncol = p)
  rcnt = 0 #p
  for (i in 1:p) {
    rcnt = rcnt + 1
    ccnt = 0 #p
    for (j in 1:p) {
      ccnt = ccnt+1
      H_mumu[rcnt,ccnt] = n*sigma_inv[i,j]
    }
  }

  #H_mumu

  #dmudsigma
  #A = sigma_inv %*% s_x_mu
  H_musigma = matrix(0,nrow = p*(p+1)/2, ncol = p)

  ccnt = 0
  for (k in 1:p){
    rcnt = 0
    ccnt = ccnt + 1
    for (i in 1:p) {

      for (j in 1:i) {

        rcnt = rcnt + 1

        if (i == j){
          #H_musigma[rcnt,ccnt] = -sigma_inv[k,i] * A[i]
          H_musigma[rcnt,ccnt] = 0
        }

        else if (i != j){
          #H_musigma[rcnt,ccnt] = -(sigma_inv[k,i] * A[j] + sigma_inv[k,j] * A[i])
          H_musigma[rcnt,ccnt] = 0
        }

      }
```

```
    }
  }

  #H_musigma

  #dsigmadsigma
  #A = n*(sigma_inv/2 - (sigma_inv %*% (c_mu/n) %*% sigma_inv))
  #A = n*((diag(p)/2 - sigma_inv %*% (c_mu/n)) %*% sigma_inv)
  A = n*sigma_inv/2
  H_sigmasigma = matrix(0,nrow =p*(p+1)/2, ncol = p*(p+1)/2)

  rcnt = 0
  for (i in 1:p) {

    for (j in 1:i) {

      rcnt = rcnt + 1
      ccnt = 0
      for (k in 1:p) {

        for (l in 1:k) {
          ccnt = ccnt + 1

          if (i==j & k==l){

            H_sigmasigma[rcnt,ccnt] = A[i,k]*sigma_inv[l,j]
            #H_sigmasigma[rcnt,ccnt] = A[k,i]*sigma_inv[j,l]

          }

          else if (i!=j & k==l){

            H_sigmasigma[rcnt,ccnt] = A[i,k]*sigma_inv[l,j] + A[j,k]*sigma_inv[l,i]
            #H_sigmasigma[rcnt,ccnt] = A[k,i]*sigma_inv[j,l] + A[k,j]*sigma_inv[i,l]

          }

          else if (i==j & k!=l){

            H_sigmasigma[rcnt,ccnt] = A[i,l]*sigma_inv[k,j] + A[i,k]*sigma_inv[l,j]
            #H_sigmasigma[rcnt,ccnt] = A[l,i]*sigma_inv[j,k] + A[k,i]*sigma_inv[i,l]

          }

          else if (i!=j & k!=l){

            H_sigmasigma[rcnt,ccnt] = A[i,l]*sigma_inv[k,j] + A[i,k]*sigma_inv[l,j] + A[j,l]*sigma_inv[l
            #H_sigmasigma[rcnt,ccnt] = A[k,i]*sigma_inv[j,l] + A[l,j]*sigma_inv[i,k] + A[k,j]*sigma_inv

          }

        }
      }
```

```
    }
  }

  #H_mumu
  #H_musigma
  #H_sigmasigma

  I = rbind(cbind(H_mumu,t(H_musigma)),
            cbind(H_musigma,H_sigmasigma))
  return(I)
}
```

```
fisher <- function(data, mu_start = NULL, sigma_start = NULL,
                   maxit = 500, tolerr = 1e-6, tolgrad = 1e-5,
                   #teta_star = NULL, #convergence_power = (1+sqrt(5))/2,
                   show = NULL){

  it = 1; stop = FALSE; for_show = matrix(0,nrow = 0,ncol = 4); p = length(mu_start)
  teta_n = mu_sigma_to_teta_vec(mu_start,sigma_start, is.gradient = FALSE)$teta #starting point


  while(it <= maxit & stop == FALSE){   #core calculation

    mu_n = teta_vec_to_mu_sigma(teta_n,p=p)$mu
    sigma_n = teta_vec_to_mu_sigma(teta_n,p=p)$sigma
    f_teta_n = loglike_f(data,mu_n,sigma_n)$l #check for positive definite???? or throw error at beginn
    grad_mu_n = grad_mu_loglike_f(data,mu_n,sigma_n)
    grad_sigma_n = grad_sigma_loglike_f(data,mu_n,sigma_n)
    grad_teta_n = mu_sigma_to_teta_vec(grad_mu_n,grad_sigma_n, is.gradient = TRUE)$grad_teta
    info_inv = solve(I(data,mu_n,sigma_n))
    #hess_inv = -1

    teta_n_new = teta_n + (info_inv %*% grad_teta_n) # Steepest Ascent #dir =  -Hess_inv* grad_teta_n #

    #need sigma to be positive definite aka positive eigenvalues    or
    #pos_definite = all(diag(teta_vec_to_mu_sigma(teta_n_new,p=3)$sigma)>0)
    pos_definite = all(eigen(teta_vec_to_mu_sigma(teta_n_new,p=p)$sigma)$values>0)

    if(pos_definite){
      mu_n_new = teta_vec_to_mu_sigma(teta_n_new,p=p)$mu
      sigma_n_new = teta_vec_to_mu_sigma(teta_n_new,p=p)$sigma
      f_teta_n_new = loglike_f(data,mu_n_new,sigma_n_new)$l
      grad_mu_n_new = grad_mu_loglike_f(data,mu_n_new,sigma_n_new) #needed if not halving
      grad_sigma_n_new = grad_sigma_loglike_f(data,mu_n_new,sigma_n_new) #needed if not halving
      grad_teta_n_new = mu_sigma_to_teta_vec(grad_mu_n_new,grad_sigma_n_new, is.gradient = TRUE)$grad_te
    }

    for_show = rbind(for_show,c(it, NaN, f_teta_n, norm(grad_teta_n, type = "2")))
    halve = 0
    while ( halve <= 20 &  (pos_definite == FALSE || f_teta_n_new < f_teta_n )){

      teta_n_new = teta_n + (info_inv %*% grad_teta_n)/2^halve  # Steepest Ascent #dir = grad_teta_n #d
```

```
    #need sigma to be positive definite aka positive eigenvalues    or
    #pos_definite = all(diag(teta_vec_to_mu_sigma(teta_n_new,p=3)$sigma)>0)
    pos_definite = all(eigen(teta_vec_to_mu_sigma(teta_n_new,p=p)$sigma)$values>0)

    if(pos_definite){
      mu_n_new = teta_vec_to_mu_sigma(teta_n_new,p=p)$mu
      sigma_n_new = teta_vec_to_mu_sigma(teta_n_new,p=p)$sigma

      #f_teta_n = loglike_f(data,mu_n,sigma_n)$l
      f_teta_n_new = loglike_f(data,mu_n_new,sigma_n_new)$l

      grad_mu_n_new = grad_mu_loglike_f(data,mu_n_new,sigma_n_new)
      grad_sigma_n_new = grad_sigma_loglike_f(data,mu_n_new,sigma_n_new)
      grad_teta_n_new = mu_sigma_to_teta_vec(grad_mu_n_new,grad_sigma_n_new, is.gradient = TRUE)$grad

      L2_norm = norm(grad_teta_n_new, type = "2")
      for_show = rbind(for_show,c(it, halve, f_teta_n_new, L2_norm))
    }
    else{
      for_show = rbind(for_show,c(it, halve, NaN, NaN))
    }

    halve = halve + 1
  }

  #stop calculation #aka convergence?    #write function to check for convergence?
  mod_rel_err = max(abs(teta_n_new-teta_n)/pmax(1,abs(teta_n_new)))
  L2_norm = norm(grad_teta_n_new, type = "2") #again just because
  if (mod_rel_err<tolerr & L2_norm < tolgrad) stop = TRUE

  teta_n <- teta_n_new #next iteration
  it = it + 1
  }



#print estimates
mu_print = data.frame(`mu`=teta_vec_to_mu_sigma(teta_n_new,p=p)$mu)
sigma_print = data.frame(`sigma` = teta_vec_to_mu_sigma(teta_n_new,p=p)$sigma)
colnames(sigma_print) = c('sigma',rep('', p-1))

print(kable(list(mu_print,sigma_print),
      align = 'c',
      booktabs = TRUE,
      caption = "Estimates"
      )
      %>% kable_styling(latex_options = "HOLD_position")
      )

#print iterations
if(show == "show_2"){
  for_show = for_show[for_show[,1]==1 | for_show[,1]==2 | for_show[,1]== (it-2) | for_show[,1]== (it-
```

```
  desc = data.frame(`it`=for_show[,1],`halve`=for_show[,2],`loglikelihood`=for_show[,3],`L2_norm`=for_sh
  return(kable(desc, col.names = names(desc), align = "cccc", booktabs = TRUE, caption = 'Iterations') %


}
```

(a) [30 points] Use the data generated and your Newton's method function to estimate the parameters in $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. Start your iterative process with

$$\boldsymbol{\mu}^{(0)} = [-1.5, 1.5, 2.3]^T \text{ and } \boldsymbol{\Sigma}^{(0)} = \begin{pmatrix} 1 & 0.5 & 0.5 \\ 0.5 & 1 & 0.5 \\ 0.5 & 0.5 & 1 \end{pmatrix}.$$

```r
mu_start = matrix(c(-1.5,1.5,2.3),nrow = 3,ncol = 1)
sigma_start = matrix(c(1,0.5,0.5,0.5,1,0.5,0.5,0.5,1),nrow = 3,ncol = 3)
newton(data, mu_start, sigma_start, maxit = 500, show = "show")
```

Table 1: Estimates

mu	sigma		
-0.9915895	0.9176864	0.6112402	0.6902982
0.9938698	0.6112402	0.9727369	0.7691461
2.0319713	0.6902982	0.7691461	1.1088345

Table 2: Iterations

it	halve	loglikelihood	L2_norm
1	NaN	-838.6352	3.9e+02
1	0	NaN	NaN
1	1	NaN	NaN
1	2	NaN	NaN
1	3	NaN	NaN
1	4	-776.8361	3.6e+02
2	NaN	-776.8361	3.6e+02
2	0	-10769.1064	2.2e+06
2	1	-722.4349	5.0e+02
3	NaN	-722.4349	5.0e+02
4	NaN	-704.8749	1.8e+02
5	NaN	-699.9388	5.3e+01
6	NaN	-699.1587	9.1e+00
7	NaN	-699.1275	4.2e-01
8	NaN	-699.1274	9.8e-04

(b) [25 points] Use the data generated and your Fisher-Scoring method function to estimate the parameters in $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. Start your iterative process with

$$\boldsymbol{\mu}^{(0)} = [-1.5, 1.5, 2.3]^T \text{ and } \boldsymbol{\Sigma}^{(0)} = \begin{pmatrix} 1 & 0.5 & 0.5 \\ 0.5 & 1 & 0.5 \\ 0.5 & 0.5 & 1 \end{pmatrix}.$$

```r
#mu_start = matrix(c(-1.5,1.5,2.3),nrow = 3,ncol = 1)
#sigma_start = matrix(c(1,0.5,0.5,0.5,1,0.5,0.5,0.5,1),nrow = 3,ncol = 3)
fisher(data, mu_start, sigma_start, maxit = 500, show = "show")
```

Table 3: Estimates

mu	sigma		
-0.9915895	0.9176864	0.6112402	0.6902982
0.9938698	0.6112402	0.9727369	0.7691461
2.0319713	0.6902982	0.7691461	1.1088345

Table 4: Iterations

it	halve	loglikelihood	L2_norm
1	NaN	-838.6352	3.9e+02
2	NaN	-733.7971	8.4e+01
3	NaN	-699.1274	4.4e-13