Homework 3 - Part 3 - Math 534

Elijah Amirianfar

2024-02-23

Part a) Generate 200 data points from a trivariate normal with

$$\boldsymbol{\mu} = (-1, 1, 2)^T, \quad \boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0.7 & 0.7 \\ 0.7 & 1 & 0.7 \\ 0.7 & 0.7 & 1 \end{bmatrix}$$

using the gen() function given during the lecture and setting the seed to 2024. Print the first three rows of your 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){
  #---- 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 = c(-1,1,2)
sig = matrix(c(1,.7,.7,.7,1,.7,.7,.7,1),3,3)

data_values = gen(200,3,mu,sig,seed=2025)
data_values[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
# The below functions are used for the entire assignment.
mu_sig2teta_vec = function(mu, sig){
  # takes a mu and sigma and puts it into a theta vector
  p = length(mu)
  teta = matrix(0, nrow = (p + p*(p+1)/2), ncol = 1)
  teta[1:p] = mu
  for (i in 1:p){
    for (j in 1:i){
      p = p+1
      teta[p] = sig[i,j]
    }
  }
  teta
}

teta_vec2mu_sig = function(p,teta){
  # takes a theta vector and outputs a mu vector and sigma matrix
  mu = teta[1:p]
  sig = matrix(0, nrow = p, ncol = p)
  for (i in 1:p){
    for (j in 1:i){
      p = p+1
      sig[i,j] = teta[p]
      sig[j,i] = sig[i,j]
    }
  }
  list(mu = mu, sig = sig)
}

gradient_mu_sig = function(x,mu,sig){ # function of all our gradients
  p = dim(sig)[1]
  n = dim(x)[1]
  siginv = solve(sig)
  C = matrix(0,p,p); # initializing sum of c_mu = (xi-mu)(xi-mu)^T
  sxm = matrix(0,p,1) # initializing sum of xi-mu
  gradm = sxm; # initializing this sum is used for the gradient w.r.t. mu

  for (i in 1:n){
    xm = x[i,] - mu # gives us (xi - mu)
    sxm = sxm + xm # does the sum of (xi - mu)
    C = C + xm %*% t(xm) # this does (xi-mu)(xi-mu)^T
  } # this calculates c_mu
  gradm = siginv %*% sxm # gives us gradient of mu

  A = n*siginv - siginv%*%C%*%siginv
  grad_s = matrix(0,dim(A)[1],dim(A)[2]) # sigma matrix

  for (i in 1:dim(sig)[1]){
    grad_s[i,i] = -.5*A[i,i] # edits the diagonals for sigma_ii
  }

  for (i in 1:dim(sig)[1]-1){
    for (j in (i+1):dim(sig)[2]){
```

```r
      grad_s[i,j] = -A[i,j] # computes sigma_ij
      grad_s[j,i] = grad_s[i,j]
    }
  }

  grad_norm = norm(mu_sig2teta_vec(gradm, grad_s), type='2')

  list(gradm = gradm, grads = grad_s, grad_norm = grad_norm)
}


likemvn = function (x, mu, sig) {
  a = dim(x)
  n = a[1] # number of rows of x
  p = a[2] # number of columns of x

  siginv = solve(sig)
  C = matrix(0,p,p); # initializing sum of c_mu = (xi-mu)(xi-mu)^T
  sxm = matrix(0,p,1) # initializing sum of xi-mu

  for (i in 1:n){
    xm = x[i,] - mu # gives us (xi - mu)
    sxm = sxm + xm # does the sum of (xi - mu)
    C = C + xm %*% t(xm) # this does (xi-mu)(xi-mu)^T
  } # this calculates c_mu

  l = -(n*p*log(2*pi)+n*log(det(sig)) + sum(solve(sig)*C))/2
  # log likelihood function
  l
}

hessian = function(x,mu,sig){
  p = dim(sig)[1]
  n = dim(x)[1]
  siginv = solve(sig)

  C = matrix(0,p,1) # initializing sum of xi-mu
  for (i in 1:n){
    xm = x[i,] - mu # gives us (xi - mu)
    C = C + xm # does the sum of (xi - mu)
  }
  C_final = siginv %*% C # gives us final value of siginv times the sum

  Z = matrix(0,p,p); # initializing sum of c_mu = (xi-mu)(xi-mu)^T
  for (i in 1:n){
    xm = x[i,] - mu # gives us (xi - mu)
    Z = Z + xm %*% t(xm) # this does (xi-mu)(xi-mu)^T
  } # this calculates c_mu
  Z_final = -.5*((-n*diag(p)+ 2 * siginv %*% Z) %*% siginv)
  # gives us final value to calculate dsig\dsig

  dmu_dmu_matrix = matrix(0,length(mu),length(mu)) # sets up matrix for dmu\dmu
  for (i in 1:p){
```

```
  for (j in 1:i){
    dmu_dmu_matrix[i,j] = -n*siginv[i,j]
    dmu_dmu_matrix[j,i] = dmu_dmu_matrix[i,j]
    # takes care of the dmu_i dmu_j portions
  }
}

dsig_dsig_matrix = matrix(0,nrow = p*(p+1)/2, ncol = p*(p+1)/2)
# sets up matrix for dsig\dsig
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){
          # calculates all the values we need for our dsig\dsig matrix
          dsig_dsig_matrix[rcnt,ccnt] = Z_final[k,i]%*%siginv[i,k]}
        else if (i != j & k != l){
          dsig_dsig_matrix[rcnt,ccnt] = Z_final[k,i]%*%siginv[j,l]+
                                        Z_final[l,j]%*%siginv[i,k]+
                                        Z_final[k,j]%*%siginv[i,l]+
                                        Z_final[l,i]%*%siginv[j,k]}
        else if (i != j & k == l){
          dsig_dsig_matrix[rcnt,ccnt] = Z_final[k,i]%*%siginv[j,k]+
                                        Z_final[k,j]%*%siginv[i,l]}
        else if (i == j & k != l){
          dsig_dsig_matrix[rcnt,ccnt] = Z_final[l,i]%*%siginv[i,k]+
                                        Z_final[k,i]%*%siginv[i,l]}
      }
    }
  }
}

dmu_dsig_matrix = matrix(0, nrow = p, ncol = p*(p+1)/2) # sets up matrix for dmu\ dsig

rcnt = 0
ccnt = 0
for (i in 1:p){
  rcnt = rcnt + 1
  ccnt = 0
  for (k in 1:p){
    for (l in 1:k){
      ccnt = ccnt + 1
      if (k == l){
        dmu_dsig_matrix[rcnt,ccnt] = -siginv[i,k]%*%C_final[k]
      }
      else if (k != l){
        dmu_dsig_matrix[rcnt,ccnt] = -siginv[i,k]%*%C_final[l] - siginv[i,l]%*%C_final[k]
      }
    }
```

```r
    }
  }
  H = rbind(cbind(dmu_dmu_matrix,(dmu_dsig_matrix)),cbind(t(dmu_dsig_matrix),
                                                          dsig_dsig_matrix))
  H
}

newton_method = function(x, mu, sig, maxit, tolerr=1e-7, tolgrad=1e-7){
  header = paste0("Iteration", "      Halving", "       log-likelihood",
                  "       ||gradient||")
  print(header)

  for (it in 1:maxit){
    theta = mu_sig2teta_vec(mu,sig) # theta^(0)
    a = likemvn(x,mu,sig) # calculates likelihood

    grad_mu = gradient_mu_sig(x,mu,sig)$gradm # gradient of mu
    grad_sig = gradient_mu_sig(x,mu,sig)$grads # gradient of sigma

    hess = hessian(x,mu,sig) # hessian of mu and sigma
    hess.i = solve(hess) # inverse of hessian

    norm_grad = gradient_mu_sig(x,mu,sig)$grad_norm # calculates norm of the gradient

    print(sprintf('%2.0f                 --          %3.4f               %.1e',
                  it,  a, norm_grad))

    direction = -1*(hess.i %*% mu_sig2teta_vec(grad_mu,grad_sig)) # calculates direction

    theta_new = theta + direction # new theta

    mu_new = teta_vec2mu_sig(length(mu), theta_new)$mu # this gives us the new mu
    sig_new = teta_vec2mu_sig(length(mu), theta_new)$sig # this gives us the new sigma
    pos_def = all(eigen(sig_new)$values > 0)
    norm_grad_new = gradient_mu_sig(x, mu_new, sig_new)$grad_norm

    if (pos_def) {atmp = likemvn(x,mu_new,sig_new)}
    else {atmp = -Inf}

    halve = 0

    print(sprintf('%2.0f                 %2.0f          %3.4f               %.1e',
                  it,  halve, atmp, norm_grad_new))
    while ((pos_def == FALSE & halve < 20) || atmp < a){
      halve = halve + 1
      theta_new = theta + direction/(2^halve)  # Newton's Method
      mu_new = teta_vec2mu_sig(length(mu), theta_new)$mu # this gives us the new mu
      sig_new = teta_vec2mu_sig(length(mu), theta_new)$sig # this gives us the new sigma
      pos_def = all(eigen(sig_new)$values > 0)
      if (pos_def) {atmp = likemvn(x,mu_new,sig_new)}
      else {atmp = -Inf}
      norm_grad_new = gradient_mu_sig(x, mu_new, sig_new)$grad_norm
```

```
      print(sprintf('%2.0f                 %2.0f          %3.4f              %.1e',
                    it,  halve, atmp, norm_grad_new))
    }

    print("-----------------------------------------------------------------")
    print(header)

    theta = theta_new
    mod_rel_error = max(abs(theta - theta_new)/abs(pmax(1,abs(theta))))
    if (mod_rel_error < tolerr & norm_grad_new < tolgrad) {
      break}
    mu = mu_new
    sig = sig_new
  }
  return(list("estimator of mu"=mu, "estimator of sigma" = sig, "iteration" = it))
}

fisher_info = function(x,mu,sig){
  p = dim(sig)[1]
  n = dim(x)[1]
  siginv = solve(sig)

  dmu_dmu_matrix = matrix(0,length(mu),length(mu)) # sets up matrix for dmu\dmu
  for (i in 1:p){
    for (j in 1:i){
      dmu_dmu_matrix[i,j] = n*siginv[i,j]
      dmu_dmu_matrix[j,i] = dmu_dmu_matrix[i,j] # takes care of the dmu_i dmu_j portions
    }
  }

  dsig_dsig_matrix = matrix(0,nrow = p*(p+1)/2, ncol = p*(p+1)/2)
  # sets up matrix for dsig\dsig
  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){
            # calculates all the values we need for our dsig\dsig matrix
            dsig_dsig_matrix[rcnt,ccnt] = (n/2)*siginv[k,i]%*%siginv[i,k]}
          else if (i != j & k != l){
            dsig_dsig_matrix[rcnt,ccnt] = (n/2)*(siginv[k,i]%*%siginv[j,l]+
                                                 siginv[l,j]%*%siginv[i,k]+
                                                 siginv[k,j]%*%siginv[i,l]+
                                                 siginv[l,i]%*%siginv[j,k])}
          else if (i != j & k == l){
            dsig_dsig_matrix[rcnt,ccnt] = (n/2)*(siginv[k,i]%*%siginv[j,k]+
                                                 siginv[k,j]%*%siginv[i,l])}
          else if (i == j & k != l){
            dsig_dsig_matrix[rcnt,ccnt] = (n/2)*(siginv[l,i]%*%siginv[i,k]+
```

```
                                                    siginv[k,i]%*%siginv[i,l])}
        }
      }
    }
  }

  dmu_dsig_matrix = matrix(0, nrow = p, ncol = p*(p+1)/2)
  # sets up matrix for dmu\ dsig

  H = rbind(cbind(dmu_dmu_matrix,(dmu_dsig_matrix)),
            cbind(t(dmu_dsig_matrix),dsig_dsig_matrix))
  H
}

fisher_method = function(x, mu, sig, maxit, tolerr=1e-7, tolgrad=1e-7){
  header = paste0("Iteration", "      Halving", "       log-likelihood",
                  "       ||gradient||")
  print(header)

  for (it in 1:maxit){
    theta = mu_sig2teta_vec(mu,sig) # theta^(0)
    a = likemvn(x,mu,sig) # calculates likelihood

    grad_mu = gradient_mu_sig(x,mu,sig)$gradm # gradient of mu
    grad_sig = gradient_mu_sig(x,mu,sig)$grads # gradient of sigma

    fisher = fisher_info(x,mu,sig) # hessian of mu and sigma
    fisher.i = solve(fisher) # inverse of hessian

    norm_grad = gradient_mu_sig(x,mu,sig)$grad_norm # calculates norm of the gradient

    print(sprintf('%2.0f                 --          %3.4f                %.1e',
                  it,  a, norm_grad))

    direction = fisher.i %*% mu_sig2teta_vec(grad_mu,grad_sig) # calculates direction

    theta_new = theta + direction # new theta

    mu_new = teta_vec2mu_sig(length(mu), theta_new)$mu # this gives us the new mu
    sig_new = teta_vec2mu_sig(length(mu), theta_new)$sig # this gives us the new sigma
    pos_def = all(eigen(sig_new)$values > 0)
    norm_grad_new = gradient_mu_sig(x, mu_new, sig_new)$grad_norm

    if (pos_def) {atmp = likemvn(x,mu_new,sig_new)}
    else {atmp = -Inf}

    halve = 0

    print(sprintf('%2.0f                 %2.0f          %3.4f                %.1e',
                  it,  halve, atmp, norm_grad_new))
    while (atmp < a || (pos_def == FALSE & halve < 20)){
      halve = halve + 1
      theta_new = theta + direction/(2^halve)  # Newton's Method
```

```
      mu_new = teta_vec2mu_sig(length(mu), theta_new)$mu # this gives us the new mu
      sig_new = teta_vec2mu_sig(length(mu), theta_new)$sig # this gives us the new sigma
      pos_def = all(eigen(sig_new)$values > 0)
      if (pos_def) {atmp = likemvn(x,mu_new,sig_new)}
      else {atmp = -Inf}
      norm_grad_new = gradient_mu_sig(x, mu_new, sig_new)$grad_norm

      print(sprintf('%2.0f                 %2.0f           %3.4f               %.1e',
                    it,  halve, atmp, norm_grad_new))
    }

    print("------------------------------------------------------------------")
    print(header)

    theta = theta_new
    mod_rel_error = max(abs(theta - theta_new)/abs(pmax(1,abs(theta))))
    if (mod_rel_error < tolerr & norm_grad_new < tolgrad) {
      break}
    mu = mu_new
    sig = sig_new
  }
  return(list("estimator of mu"=mu, "estimator of sigma" = sig, "iteration" = it))
}
```

Part b) Use the data you generated in part (a) and your Newton's method function with step-halving to estimate the parameters in $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. Start your iterative process with the following:

$$\boldsymbol{\mu}^{(0)} = (-1.5, 1.5, 2.3)^T, \quad \boldsymbol{\Sigma}^{(0)} = \begin{pmatrix} 1 & 0.5 & 0.5 \\ 0.5 & 1 & 0.5 \\ 0.5 & 0.5 & 1 \end{pmatrix}, \quad maxit = 500, \quad tolerr = 1e\text{-}7, \quad tolgrad = 1e\text{-}7$$

```
mu = matrix(c(-1.5,1.5,2.3),3,1)
sig = matrix(c(1,.5,.5,.5,1,.5,.5,.5,1),3,3)

newton_method(data_values,mu,sig,500)
```

```
## [1] "Iteration      Halving       log-likelihood       ||gradient||"
## [1] " 1               --           -838.6352               3.9e+02"
## [1] " 1               0            -Inf               7.0e+02"
## [1] " 1               1            -Inf               1.1e+03"
## [1] " 1               2            -Inf               2.6e+05"
## [1] " 1               3            -Inf               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 c) Use the data generated in part (a) and your Fisher-scoring method function with step-halving to estimate the parameters in $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. Start your iterative process with the following:

$$\boldsymbol{\mu}^{(0)} = (-1.5, 1.5, 2.3)^T, \quad \boldsymbol{\Sigma}^{(0)} = \begin{pmatrix} 1 & 0.5 & 0.5 \\ 0.5 & 1 & 0.5 \\ 0.5 & 0.5 & 1 \end{pmatrix}, \quad maxit = 500, \quad tolerr = 1e\text{-}7, \quad tolgrad = 1e\text{-}7$$

```
mu = matrix(c(-1.5,1.5,2.3),3,1)
sig = matrix(c(1,.5,.5,.5,1,.5,.5,.5,1),3,3)
fisher_method(data_values,mu,sig,500)
```

```
## [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              4.3e-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
```