Math 534 Homework 3.4

Mike Palmer

due 2024/02/28

Data Generation To get a dataset, use gen() function with seed 2025 to generate 200 data points from a trivariate normal with the following parameters.

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

```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
```

Exercise J-2.2 (continued) [20 Points] 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 $\boldsymbol{\Sigma} = (\sigma_{ij})$.

You are to use the BFGS quasi-Newton method in the R optim 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

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

```
#for kable but not used here -> #results = 'asis'

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_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
}

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
```

```r
      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]
        }
      }
    }
  }

  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)
}

#new code to run optim()

f_optim <- function(teta,data){
  p = ncol(data)
  mu = teta_vec_to_mu_sigma(teta,p)$mu
  sigma = teta_vec_to_mu_sigma(teta,p)$sigma
  pos_definite = all(eigen(sigma)$values>0)
  if (pos_definite){l = loglike_f(data,mu,sigma)$l} #crucial point here
  else {l= NaN}
  l
}

gr_optim <- function(teta,data){
  p = ncol(data)
  mu = teta_vec_to_mu_sigma(teta,p)$mu
  sigma = teta_vec_to_mu_sigma(teta,p)$sigma
  grad_mu = grad_mu_loglike_f(data,mu,sigma)
  grad_sigma = grad_sigma_loglike_f(data,mu,sigma)
  grad = mu_sigma_to_teta_vec(grad_mu,grad_sigma, is.gradient = TRUE)$grad_teta
```

```r
  grad
}

mu_start = matrix(c(0,0,0),nrow = 3,ncol = 1)
sigma_start = diag(3)
teta_start = mu_sigma_to_teta_vec(mu_start,sigma_start, is.gradient = FALSE)$teta


optim(teta_start, f_optim, gr_optim, data = data, method = "BFGS",
      control = list(fnscale = -1, trace = 1, abstol = 10e-6), hessian = TRUE)
```

```
## initial  value 1461.282329
## iter  10 value 740.079678
## iter  20 value 699.166236
## iter  30 value 699.128054
## final  value 699.127438
## converged
```

```
## $par
##             [,1]
##  [1,] -0.9915896
##  [2,]  0.9938697
##  [3,]  2.0319712
##  [4,]  0.9176866
##  [5,]  0.6112404
##  [6,]  0.9727371
##  [7,]  0.6902985
##  [8,]  0.7691464
##  [9,]  1.1088348
##
## $value
## [1] -699.1274
##
## $counts
## function gradient
##      111       31
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##                [,1]          [,2]          [,3]          [,4]          [,5]
##  [1,] -4.464025e+02  1.345794e+02  1.845538e+02 -9.301890e-06 -1.432141e-05
##  [2,]  1.345794e+02 -4.959286e+02  2.602207e+02  2.804305e-06 -5.171009e-06
##  [3,]  1.845538e+02  2.602207e+02 -4.757652e+02  3.845678e-06  1.250252e-05
##  [4,] -9.301890e-06  2.804305e-06  3.845678e-06 -4.981954e+02  3.003905e+02
##  [5,] -1.432141e-05 -5.171009e-06  1.250252e-05  3.003905e+02 -1.197504e+03
##  [6,]  5.162979e-06 -1.902547e-05  9.982990e-06 -4.527977e+01  3.337179e+02
##  [7,] -2.927389e-05  1.540709e-05  3.778654e-06  4.119372e+02  4.566367e+02
##  [8,]  1.706486e-05 -2.681124e-05  1.054376e-06 -1.241868e+02  2.825287e+02
##  [9,]  1.369243e-05  1.930640e-05 -3.529804e-05 -8.515167e+01 -2.401267e+02
```

```
##                [,6]          [,7]          [,8]          [,9]
##  [1,]  5.162979e-06 -2.927389e-05  1.706486e-05  1.369243e-05
##  [2,] -1.902547e-05  1.540709e-05 -2.681124e-05  1.930640e-05
##  [3,]  9.982990e-06  3.778654e-06  1.054376e-06 -3.529804e-05
##  [4,] -4.527977e+01  4.119372e+02 -1.241868e+02 -8.515167e+01
##  [5,]  3.337179e+02  4.566367e+02  2.825287e+02 -2.401267e+02
##  [6,] -6.148743e+02 -1.751036e+02  6.452751e+02 -1.692901e+02
##  [7,] -1.751036e+02 -1.232248e+03  8.001323e+01  4.390333e+02
##  [8,]  6.452751e+02  8.001323e+01 -1.518361e+03  6.190392e+02
##  [9,] -1.692901e+02  4.390333e+02  6.190392e+02 -5.658909e+02
```

Exercise GH-2.3

(a) [5 Points]

Latex showing likelihood here.

(b) [20 Points]

```
###########################################################data
#censored #treatment #w_i = 0 #d_i = 1
t_i = c(6,9,10,11,17,19,20,25,32,32,34,35)
w_i = c(rep(0,12))
d_i = c(rep(1,12))

#uncensored #treatment  #w_i = 1 #d_i = 1
t_i = c(t_i,6,6,6,7,10,13,16,22,23)
w_i = c(w_i,rep(1,9))
d_i = c(d_i,rep(1,9))

#uncensored #control #w_i = 1 #d_i = 0
t_i = c(t_i,1,1,2,2,3,4,4,5,5,8,8,8,8,11,11,12,12,15,17,22,23)
w_i = c(w_i,rep(1,21))
d_i = c(d_i,rep(0,21))

data<-cbind(d_i,w_i,t_i)
#length(t_i)

loglike <- function(data,alpha,beta0,beta1){
  l = 0
  for(i in 1:nrow(data)){
  d_i = data[[i,1]]
  w_i = data[[i,2]]
  t_i = data[[i,3]]
  l = l + w_i*log(alpha)+w_i*(alpha-1)*log(t_i)-(t_i^(alpha))*exp(beta0+d_i*beta1)
  }
  l
}

grad_loglike <- function(data,alpha,beta0,beta1){
  d_alpha = 0
  d_beta0 = 0
  d_beta1 = 0
  for(i in 1:nrow(data)){
    d_i = data[[i,1]]
    w_i = data[[i,2]]
    t_i = data[[i,3]]
    d_alpha = d_alpha + w_i/alpha+w_i*log(t_i)-(t_i^(alpha))*log(t_i)*exp(beta0+d_i*beta1)
    d_beta0 = d_beta0 + -(t_i^(alpha))*exp(beta0+d_i*beta1)
    d_beta1 = d_beta1 + -(t_i^(alpha))*exp(beta0+d_i*beta1)*d_i
  }
  return(matrix(c(d_alpha,d_beta0,d_beta1),nrow=3,ncol=1))
}

hess_loglike <- function(data,alpha,beta0,beta1){
  dd_alphaalpha =  0
  dd_beta0beta0 =  0
```

```r
  dd_beta1beta1 =  0
  dd_alphabeta0 =  0
  dd_alphabeta1 =  0
  dd_beta0beta1 =  0

  for(i in 1:nrow(data)){
    d_i = data[[i,1]]
    w_i = data[[i,2]]
    t_i = data[[i,3]]
    dd_alphaalpha = dd_alphaalpha + -w_i/(alpha^2)-2*(t_i^(alpha))*log(t_i)*exp(beta0+d_i*beta1)
    dd_beta0beta0 = dd_beta0beta0 + -(t_i^(alpha))*exp(beta0+d_i*beta1)
    dd_beta1beta1 = dd_beta1beta1 + -(t_i^(alpha))*exp(beta0+d_i*beta1)*d_i^2
    dd_alphabeta0 = dd_alphabeta0 + -(t_i^(alpha))*log(t_i)*exp(beta0+d_i*beta1)
    dd_alphabeta1 = dd_alphabeta1 + -(t_i^(alpha))*log(t_i)*exp(beta0+d_i*beta1)*d_i
    dd_beta0beta1 = dd_beta0beta1 + -(t_i^(alpha))*exp(beta0+d_i*beta1)*d_i
  }
  H = matrix(c(dd_alphaalpha,dd_alphabeta0,dd_alphabeta1,
               dd_alphabeta0,dd_beta0beta0,dd_beta0beta1,
               dd_alphabeta1,dd_beta0beta1,dd_beta1beta1),nrow =3, ncol =3)
  return(H)
}


#-solve(Hess)*grad
#alpha=1;beta0=1;beta1=1
#loglike(data,alpha,beta0,beta1)
#grad_loglike(data,alpha,beta0,beta1)
#hess_loglike(data,alpha,beta0,beta1)


newton <- function(data, alpha_start=1,beta0_start=1,beta1_start=1,
                   maxit = 500, tolerr = 1e-2, tolgrad = 1e-2,
                   #teta_star = NULL, #convergence_power = (1+sqrt(5))/2,
                   show = NULL){

  it = 1; stop = FALSE; for_show = matrix(0,nrow = 0,ncol = 4); p = 3
  teta_n = matrix(c(alpha_start,beta0_start,beta1_start),nrow=3,ncol=1) #starting point


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

    alpha = teta_n[1,]
    beta0 = teta_n[2,]
    beta1 = teta_n[3,]
    f_teta_n = loglike(data,alpha,beta0,beta1)
    grad_teta_n = grad_loglike(data,alpha,beta0,beta1)
    hess_inv = solve(hess_loglike(data,alpha,beta0,beta1))

    teta_n_new = teta_n + (-hess_inv %*% grad_teta_n)

    if(teta_n_new[1,]>0){
      alpha = teta_n_new[1,]
      beta0 = teta_n_new[2,]
```

```r
      beta1 = teta_n_new[3,]
      f_teta_n_new = loglike(data,alpha,beta0,beta1)
      grad_teta_n_new = grad_loglike(data,alpha,beta0,beta1)
    }

    for_show = rbind(for_show,c(it, NaN, f_teta_n, norm(grad_teta_n, type = "2")))
    halve = 0
    while( halve <= 20 & (teta_n_new[1,]<=0 || 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 #

      if(teta_n_new[1,]>0){
        alpha = teta_n_new[1,]
        beta0 = teta_n_new[2,]
        beta1 = teta_n_new[3,]
        f_teta_n_new = loglike(data,alpha,beta0,beta1)
        grad_teta_n_new = grad_loglike(data,alpha,beta0,beta1)

        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
  parameter_print = data.frame(`alpha`=teta_n_new[1,],
                               `beta0`=teta_n_new[2,],
                               `beta1`=teta_n_new[3,])

  print(kable(list(parameter_print),
              align = 'c',
              booktabs = TRUE,
              caption = "Parameter 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') %

}
```

```r
newton(data, 154,-490,-60,maxit = 1000, show = "show_2")
```

Table 1: Parameter Estimates

alpha	beta0	beta1
153.8792	-489.6221	-59.94931

Table 2: Iterations

it	halve	loglikelihood	L2_norm
1	NaN	9256.877	5.9e+01
1	0	1668.795	6.0e+01
1	1	5472.874	6.0e+01
1	2	7365.918	6.0e+01
1	3	8311.597	5.9e+01
1	4	8784.281	5.9e+01
1	5	9020.590	5.9e+01
1	6	9138.736	5.9e+01
1	7	9197.807	5.9e+01
1	8	9227.343	5.9e+01
1	9	9242.110	5.9e+01
1	10	9249.494	5.9e+01
1	11	9253.186	5.9e+01
1	12	9255.031	5.9e+01
1	13	9255.954	5.9e+01
1	14	9256.416	5.9e+01
1	15	9256.647	5.9e+01
1	16	9256.762	5.9e+01
1	17	9256.820	5.9e+01
1	18	9256.849	5.9e+01
1	19	9256.863	5.9e+01
1	20	9256.870	5.9e+01
2	NaN	9256.870	5.9e+01
2	0	1668.780	6.0e+01
2	1	5472.863	6.0e+01
2	2	7365.909	6.0e+01
2	3	8311.588	5.9e+01
2	4	8784.274	5.9e+01
2	5	9020.582	5.9e+01
2	6	9138.729	5.9e+01
2	7	9197.800	5.9e+01
2	8	9227.335	5.9e+01
2	9	9242.103	5.9e+01
2	10	9249.487	5.9e+01
2	11	9253.178	5.9e+01
2	12	9255.024	5.9e+01

2	13	9255.947	5.9e+01
2	14	9256.409	5.9e+01
2	15	9256.639	5.9e+01
2	16	9256.755	5.9e+01
2	17	9256.813	5.9e+01
2	18	9256.841	5.9e+01
2	19	9256.856	5.9e+01
2	20	9256.863	5.9e+01
999	NaN	9249.678	5.9e+01
999	0	1654.097	6.0e+01
999	1	5462.011	6.0e+01
999	2	7356.891	6.0e+01
999	3	8303.484	5.9e+01
999	4	8776.625	5.9e+01
999	5	9013.162	5.9e+01
999	6	9131.422	5.9e+01
999	7	9190.551	5.9e+01
999	8	9220.114	5.9e+01
999	9	9234.896	5.9e+01
999	10	9242.287	5.9e+01
999	11	9245.982	5.9e+01
999	12	9247.830	5.9e+01
999	13	9248.754	5.9e+01
999	14	9249.216	5.9e+01
999	15	9249.447	5.9e+01
999	16	9249.562	5.9e+01
999	17	9249.620	5.9e+01
999	18	9249.649	5.9e+01
999	19	9249.663	5.9e+01
999	20	9249.671	5.9e+01
1000	NaN	9249.671	5.9e+01
1000	0	1654.082	6.0e+01
1000	1	5462.000	6.0e+01
1000	2	7356.882	6.0e+01
1000	3	8303.476	5.9e+01
1000	4	8776.618	5.9e+01
1000	5	9013.155	5.9e+01
1000	6	9131.415	5.9e+01
1000	7	9190.543	5.9e+01
1000	8	9220.107	5.9e+01
1000	9	9234.889	5.9e+01
1000	10	9242.280	5.9e+01
1000	11	9245.975	5.9e+01
1000	12	9247.823	5.9e+01
1000	13	9248.747	5.9e+01
1000	14	9249.209	5.9e+01
1000	15	9249.440	5.9e+01
1000	16	9249.555	5.9e+01
1000	17	9249.613	5.9e+01
1000	18	9249.642	5.9e+01

1000	19	9249.656	5.9e+01
1000	20	9249.663	5.9e+01

(d) [5 Points]

```
hess_loglike(data, 154,-490,-60)
```

```
##             [,1]        [,2]        [,3]
## [1,] -0.6107199 -0.30472748 -0.30222396
## [2,] -0.3047275 -0.08581180 -0.08501334
## [3,] -0.3022240 -0.08501334 -0.08501334
```

(d using optim)

```
teta = c(154,-490,-60)

fn <- function(teta,data){
  alpha = teta[1]
  beta0 = teta[2]
  beta1 = teta[3]
  if (TRUE){ l = loglike(data,alpha,beta0,beta1)}
  else {l = NaN}
  l
}

gn <- function(teta,data){
  alpha = teta[1]
  beta0 = teta[2]
  beta1 = teta[3]
  grad = grad_loglike(data,alpha,beta0,beta1)
  grad
}

#fn(teta,data)
#gn(teta,data)


optim(teta,fn,gn, data = data, method = "BFGS", control = list(fnscale = -1, trace = 1, abstol = 10e-6))
```

```
## initial  value -9256.877409
## final  value -9284.876564
## converged
```

```
## $par
## [1]  154.47524 -490.00069  -60.00068
##
## $value
## [1] 9284.877
##
## $counts
## function gradient
##        5        1
##
## $convergence
## [1] 0
##
```

```
## $message
## NULL
##
## $hessian
##           [,1]       [,2]       [,3]
## [1,] -5.847998 -1.6459500 -1.6348481
## [2,] -1.645950 -0.4634103 -0.4598696
## [3,] -1.634848 -0.4598696 -0.4598696
```

```
ans<-optim(teta,fn,gn, data = data, method = "BFGS", control = list(fnscale = -1, trace = 1, abstol = 1
```

```
## initial  value -9256.877409
## final  value -9284.876564
## converged
```

```
hess<-optim(teta,fn,gn, data = data, method = "BFGS", control = list(fnscale = -1, trace = 1, abstol =
```

```
## initial  value -9256.877409
## final  value -9284.876564
## converged
```

```
#optim(teta,fn,gn, data = data, method = "L-BFGS-B",lower = c(1,-200,-200), upper = c(1400,1400,1400),

solve(hess_loglike(data,ans[[1]],ans[[2]],ans[[3]]))
```

```
##            [,1]        [,2]         [,3]
## [1,]  0.3916085   -1.227868   -0.1643089
## [2,] -1.2278680 -278.574757  282.9398553
## [3,] -0.1643089  282.939855 -284.5302636
```

```
#sqrt(-diag(solve(hess)))
```