Math 534 Homework 3.2 - 30 points

Mike Palmer
due 2024/02/21

Exercise J-2.2 Write a general function to maximize the following log-likelihood function with respect to parameters $\boldsymbol{\mu} = [\mu_1, \mu_2, \ldots, \mu_p]^T$ and $\boldsymbol{\Sigma} = (\sigma_{ij})$:

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

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 the Steepest ascent method with step-halving to obtain the maximum likelihood estimate of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ for a given set of $n \times p$ matrix of data.

```
library(kableExtra) #for output styling
library(dplyr) #for output styling
```

```
#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)
}

#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
```

```r
    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]
                  }
              }
    }
  }

  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)
}

#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

steepest_ascent <- function(data, mu_start = NULL, sigma_start = NULL, teta_start = NULL, p = NULL,
                            maxit = 500, tolerr = 1e-6, tolgrad = 1e-5,
                            #teta_star = NULL, #convergence_power = (1+sqrt(5))/2,
                            show_2 = FALSE, return_estimates = FALSE){

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

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

    mu_n = teta_vec_to_mu_sigma(teta_n,p=p)$mu
```

```r
  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

  teta_n_new = teta_n + grad_teta_n # Steepest Ascent #dir = grad_teta_n #dir for direction

  #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
  }

  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 + grad_teta_n/2^halve  # Steepest Ascent #dir = grad_teta_n #dir for directio

    #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

      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
      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_sigma_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
  if(show_2 == TRUE){
    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_
      mutate(it,halve,loglikelihood,L2_norm = sprintf('%4.1e',L2_norm))
    final = kable(desc, col.names = names(desc),align = "ccc")
    return(final)
  }
  else{
    desc = data.frame(`it`=for_show[,1],`halve`=for_show[,2],`loglikelihood`=for_show[,3],`L2_norm`=for_
      mutate(it,halve,loglikelihood,L2_norm = sprintf('%4.1e',L2_norm))
    return(desc)
  }
  if(return_estimates == TRUE)
    return(list("mu estimate" = teta_vec_to_mu_sigma(teta_n_new,p=3)$mu,
                "sigma estimate" = teta_vec_to_mu_sigma(teta_n_new,p=3)$sigma))
}
```

(a) [5 points] Generate 200 data points from a trivariate normal with

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

```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 = 2024)
data[1:3,]
```

```
##            [,1]      [,2]     [,3]
## [1,]  0.5341745 1.9975269 4.092011
## [2,] -0.1649303 1.8387117 3.010171
## [3,] -1.2914162 0.3417351 1.871737
```

(b) [25 points]

```r
mu_start = matrix(c(0,0,0),nrow = 3,ncol = 1)
sigma_start = matrix(c(1,0,0,0,1,0,0,0,1),nrow = 3,ncol = 3)

steepest_ascent(data, mu_start = mu_start, sigma_start = sigma_start,
                maxit = 500, show_2 = TRUE, return_estimates = FALSE)
```

it	halve	loglikelihood	L2_norm
1	NaN	-1446.8510	9.1e+02
1	0	NaN	NaN
1	1	NaN	NaN
1	2	NaN	NaN
1	3	NaN	NaN
1	4	NaN	NaN
1	5	NaN	NaN
1	6	NaN	NaN
1	7	NaN	NaN
1	8	NaN	NaN
1	9	-899.7795	1.9e+02
2	NaN	-899.7795	1.9e+02
2	0	NaN	NaN
2	1	NaN	NaN
2	2	NaN	NaN
2	3	NaN	NaN
2	4	NaN	NaN
2	5	NaN	NaN
2	6	NaN	NaN
2	7	NaN	NaN
2	8	-895.1564	9.2e+02
457	NaN	-682.1694	1.7e-05
457	0	-682.1694	3.0e-02
457	1	-682.1694	1.5e-02
457	2	-682.1694	7.4e-03
457	3	-682.1694	3.7e-03
457	4	-682.1694	1.8e-03
457	5	-682.1694	9.1e-04
457	6	-682.1694	4.5e-04
457	7	-682.1694	2.2e-04
457	8	-682.1694	1.0e-04
457	9	-682.1694	4.8e-05
458	NaN	-682.1694	4.8e-05
458	0	-682.1694	1.2e-01
458	1	-682.1694	5.9e-02
458	2	-682.1694	3.0e-02
458	3	-682.1694	1.5e-02
458	4	-682.1694	7.4e-03
458	5	-682.1694	3.7e-03
458	6	-682.1694	1.8e-03
458	7	-682.1694	8.8e-04
458	8	-682.1694	4.2e-04
458	9	-682.1694	1.9e-04
458	10	-682.1694	7.0e-05
458	11	-682.1694	1.6e-05