Report_HW3P3

Johnny Martinez

3/23/2022

Exercise J-2.2

Helper functions to use for optim()

- Functions:

1. param_convert() - combines mu and sigma into one vector and can breakdown theta vector into mu and sigma
2. diff_xi_mu() - computes xi-mu and C := sum of (xi-mu)(xi-mu)^T
3. log_like_mvn() - computes the log-likelihood for a multivariate normal
4. gradient() - computes the gradient

```
param_convert <- function(mu=NULL, sigma=NULL, theta=NULL, t_comp=FALSE, ms_comp=FALSE){
    # decompose mu and sig into theta vector
    if(t_comp){t <- matrix(c(mu, sigma[upper.tri(sigma, TRUE)]),ncol=1)}
    # decompose theta vector into mu and sig
    if(ms_comp){
        p <- sqrt(2*length(theta) + 9/4) - 3/2 # formula derived from num of thetas = p + p(p+1)/2
        mu <- matrix(theta[1:p],p,1) # create new mu vector

        sig <- matrix(0,p,p) # create new sig matrix
        sig[upper.tri(sig,TRUE)] <- theta[-c(1:p)]
        sig[lower.tri(sig)] <- t(sig)[lower.tri(sig)]
    }
    #writeLines("leave param_conv function")
    list(t=if(t_comp){t}, mu=if(ms_comp){mu}, sig=if(ms_comp){sig})
}

diff_xi_mu <- function(mu, comp_C=FALSE, comp_sxm=FALSE){
    # Compute C defined as sum of (xi-mu)(xi-mu)^T
    xm <- apply(datan, 1, function(x) x-mu) # subtract mu from rows of x
    C <- xm %*% t(xm)

    # Compute sum of (xi-mu) for use in gradient
    if(comp_sxm){
        sxm <- matrix(rowSums(xm), nrow=ncol(datan),1)
    }

    list(C = if(comp_C) C, sxm = if(comp_sxm) sxm)
}
```

```r
log_like_mvn <- function(theta){
    n <- nrow(datan) # num of rows
    p <- ncol(datan) # num of columns

    t <- param_convert(theta=theta, ms_comp=TRUE)
    sigma <- t$sig
    mu <- t$mu

    if(any(eigen(sigma)$values <= 0)){
        sigma[,] <- -Inf
    }

    C <- diff_xi_mu(mu, comp_C=TRUE)$C # C := sum of (xi-mu)(xi-mu)^T

    log_det_sig <- log(det(sigma)) # log determinant of sigma
    sig_inv <- solve(sigma) # sigma inverse

    # Compute log likelihood function for multivariate normal
    # sum(sig.inv * C) = trace(sig.inv %*% C)
    log_like <- (-1/2)*(n*p*log(2*pi)+n*log_det_sig + sum(sig_inv * C ))
    log_like

}

gradient <- function(theta, t_comp=TRUE, dmu_comp=FALSE, dsig_comp=FALSE){
    t <- param_convert(theta=theta, ms_comp=TRUE)
    mu <- t$mu
    sigma <- t$sig

    if(any(eigen(sigma)$values <= 0)){
        sigma[,] <- -Inf
    }

    #writeLines("\n entered in gradient function")
    sig_inv <- solve(sigma)
    diff <- diff_xi_mu(mu, comp_C=TRUE, comp_sxm=TRUE)
    sxm <- diff$sxm
    C <- diff$C

    # partial deriv_ formulas for mu indices
    dmu <- sig_inv %*% sxm

    # partial deriv_ formulas for sigma indices
    off_d_mat <- -(n * sig_inv - sig_inv %*% C %*% sig_inv) # take off diag elem_
    diag_mat <- (-1/2)*(n*sig_inv - sig_inv %*% C %*% sig_inv) # take diag elem_

    # dsig
    dsig <- matrix(0, nrow=p, ncol=p) # initialize pxp matrix for gradient of sigmas
    dsig[upper.tri(dsig)] <- off_d_mat[upper.tri(off_d_mat)] # fill matrix with lower tri of off_d_mat
    dsig[lower.tri(dsig)] <- t(dsig)[lower.tri(dsig)] # fill matrix with upper tri of off_d_mat
    diag(dsig) <- diag(diag_mat) # fill matrix with diag elements diagmat

    # theta form
```

```
    t <- param_convert(mu=dmu, sigma=dsig, t_comp=TRUE)$t
    t
}
```

Provided function to generate data and print first 3 rows

```
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
}

# Generate data

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 matrix
    # Seed can be optionally set for the random number generator
    set.seed(seed)
    # generate data from normal mu sigma
    x = matrix(rnorm(n*p),n,p)
    datan = x %*% sqrtm(sig) + matrix(mu,n,p, byrow = TRUE)
    datan
}

# sample data
n <- 200
p <- 3
sigma <- matrix(c(1,.7,.7, .7, 1, .7, .7, .7, 1),p,p)
mu <- matrix(c(-1,1,2),p,1)
datan = gen(n,p,mu,sigma,seed = 2022)
```

3 first rows of data

```
##            [,1]       [,2]      [,3]
## [1,]  0.1947111  1.9078866  3.153741
## [2,] -2.8172598 -0.2811456 -0.352587
## [3,] -2.4052730 -1.0723173  1.237333
```

Use optim() with gradient function and log-likelihood function from above

```
mu0 <- matrix(c(0,0,0), ncol=1)
sig0 <- diag(3)
theta0 <- param_convert(mu0, sig0, t_comp=TRUE)$t

optim(par=theta0, fn=log_like_mvn, gr=gradient, method="BFGS",
      control = list(fnscale=-1, trace=100, abstol=1e-5))
```

```
## initial  value 1449.380448
## iter  10 value 723.132199
## iter  20 value 692.631928
## iter  30 value 692.590246
## final  value 692.587107
## converged
```

```
## $par
##             [,1]
##  [1,] -1.0402718
##  [2,]  0.9439508
##  [3,]  1.9896595
##  [4,]  1.0807572
##  [5,]  0.7093068
##  [6,]  0.9189932
##  [7,]  0.7800262
##  [8,]  0.6881343
##  [9,]  1.0484688
##
## $value
## [1] -692.5871
##
## $counts
## function gradient
##      115       31
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Exercise GH-2.3:

Part (a) See attached hand-written page

Part (b)

```
library(formattable)
library(here)
data2 <- read.csv(here("data","censor_data.csv"))
```

```
log_like_weib <- function(theta){
    a = theta[1]
    b0 = theta[2]
    b1 = theta[3]
    t = data2$t
    w = data2$w
    d = data2$d
    like <- sum((w*log(t^a * exp(b0+d*b1))) - t^a * exp(b0+d*b1) + w*log(a/t))
```

```
    like
}

gradient <- function(theta){
    a = theta[1]
    b0 = theta[2]
    b1 = theta[3]
    t = data2$t
    w = data2$w
    d = data2$d

    da <- sum(w/a + w*log(t) - t^a*exp(b0 + b1*d)*log(t))
    db0 <- sum(w - t^a*exp(b0 + b1*d))
    db1 <- sum(d*w - d*t^a*exp(b0 + b1*d))
    rbind(da,db0,db1)
}

hessian <- function(theta){
    a = theta[1]
    b0 = theta[2]
    b1 = theta[3]
    t = data2$t
    w = data2$w
    d = data2$d

    hess <- matrix(0, nrow=3, ncol=3)

    daa <- sum(-w/a^2 - t^a*exp(b0 + b1*d)*log(t)^2)
    dab0 <- sum(-t^a*exp(b0 + b1*d)*log(t))
    dab1 <- sum(-d*t^a*exp(b0 + b1*d)*log(t))
    fa <- cbind(daa,dab0,dab1)

    db0b0 <- sum(-t^a*exp(b0 + b1*d))
    db0b1 <- sum(-d*t^a*exp(b0 + b1*d))
    fb0 <- cbind(dab0, db0b0, db0b1)

    db1b1 <- sum(-d^2*t^a*exp(b0 + b1*d))
    fb1 <- cbind(dab1, db0b1, db1b1)

    hess[1,] <- fa
    hess[2,] <- fb0
    hess[3,] <- fb1

    hess
}

stop.criteria <- function(t0, t1, grad, tol.mre=1e-6, tol.grad=1e-5, itr, max.itr=100){
    mre <- max(abs((t1 - t0)/pmax(1, t1)))
    gradn <- norm(grad,"2")
    stop <- ifelse(itr == max.itr | gradn < tol.grad | mre < tol.mre, TRUE, FALSE)
    list(stop=stop, norm.g=gradn)
}
```

```
newton <- function(theta, tol.grad=1e-5, tol.mre=1e-6, max.itr=50){

    it <- 1
    stop <- FALSE

    while(!stop){
        obj.fn <- log_like_weib(theta) # set objective function

        grad <- gradient(theta)
        hess <- hessian(theta)

        dir <- -solve(hess) %*% grad

        t1 <- theta + dir
        fun2 <- log_like_weib(t1)

        halve = 0
        while(fun2 < obj.fn & halve <= 20){
            halve = halve + 1
            t1 <- t0 + dir/2^halve
            fun2 <- log.like.weib(t1)
            print(c(it, halve))
        }
        if (halve >= 20) print('Step-halving failed after 20 halvings')

        obj.fn <- fun2

        stop.cri <- stop.criteria(theta, t1, grad=grad, itr=it)
        grad_norm <- stop.cri$norm.g


        stop <- stop.cri$stop

        it <- it + 1

        theta <- t1

        #print(c(it, halve,obj.fn, fun2))
        writeLines(paste("\nitr: ", it-1))
        writeLines(paste("halving   ", "log-like     ", "norm   "))
        writeLines(paste(halve, "        ", formattable(obj.fn, digits=4, format="f"), "   ", formattabl

        print ('------------------------------------------')

    }
    theta
}

# initial value
theta <- c(1,0,0)
results <- newton(theta)
```

```
##
```

```
## itr:  1
## halving    log-like      norm
## 0          -237.4147     1.6e+03
## [1] "----------------------------------------"
##
## itr:  2
## halving    log-like      norm
## 0          -139.4431     5.5e+02
## [1] "----------------------------------------"
##
## itr:  3
## halving    log-like      norm
## 0          -112.5964     1.8e+02
## [1] "----------------------------------------"
##
## itr:  4
## halving    log-like      norm
## 0          -107.1589     5.4e+01
## [1] "----------------------------------------"
##
## itr:  5
## halving    log-like      norm
## 0          -106.5914     1.2e+01
## [1] "----------------------------------------"
##
## itr:  6
## halving    log-like      norm
## 0          -106.5795     1.6e+00
## [1] "----------------------------------------"
##
## itr:  7
## halving    log-like      norm
## 0          -106.5795     3.9e-02
## [1] "----------------------------------------"
##
## itr:  8
## halving    log-like      norm
## 0          -106.5795     2.5e-05
## [1] "----------------------------------------"
```

Parameter estimation results using intial value = (1,1,1)

```
## alpha: 1.3657575
## b0: -3.0707041
## b1: -1.7308717
```

Part (d)

Compute standard errors using observed info matrix and then compute correlation matrix

```r
observed_info <- -solve(hessian(results))

sd_err <- sqrt(diag(observed_info))
```

```
## std. errors:
```

```
## [1] 0.2011650 0.5580702 0.4130819
```

```
## correlation matrix:
```

```
##            [,1]        [,2]        [,3]
## [1,]  1.0000000 -0.92038120 -0.26415291
## [2,] -0.9203812  1.00000000  0.03655683
## [3,] -0.2641529  0.03655683  1.00000000
```

- It can be seen in the correlation matrix that alpha is highly negatively correlated with b0, while there is a slight correlation between alpha and b1. Moreover, there is almost no correlation between b0 and b1.