MATH 534 HOMEWORK 5

Gustavo Esparza

3/9/2019

1

Preliminary Code

```
library(MCMCpack)    # Used for vech and xpnd to vectorize and unvectorize a matrix.
```

```
sqrtm = function (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 = 2013){

  set.seed(seed)
  x = matrix(rnorm(n*p),n,p)
  datan = x %*% sqrtm(sig) + matrix(mu,n,p, byrow = TRUE)
  datan
}
datan =gen(200, 3, c(1,-1,2), matrix(c(1,.7,.7,.7,1,.7,.7,.7,1),3) )
mu_0=c(0,0,0)
sigma_0=matrix(c(1,0,0,0,1,0,0,0,1),3)

mu_n =c(-1.5,1.5,2.3)
sigma_n=matrix(c(1,.5,.5,.5,1,.5,.5,.5,1),3)

tolerr=10e-6
tolgrad=10e-9
```

Likelihood/Gradient function:

```
likemvn = function (x,mu,sig,gcomp) {
  a = dim(x)
  n = a[1]
  p = a[2]
  siginv = solve(sig)
  C= matrix(0,p,p);
  sxm = matrix(0,p,1)
  gradm = sxm;
  for (i in 1:n){
    xm = x[i,] - mu
    sxm = sxm + xm
    C = C + xm %*% t(xm)}
  D = siginv%*%(n*sig-C)%*%siginv

  if(gcomp==TRUE){
    gradm = siginv %*% sxm
    grads= -D
    grads = grads - (1/2)*diag(diag(grads))}  #Removing double counted diagonals
```

```
  l = -(n*p*log(2*pi)+n*log(det(sig)) + sum(siginv * C ))/2
  list(l = l, gradm = if(gcomp) gradm, grads = if(gcomp) grads) }
```

HESSIAN function:

```
hessian = function(x,mu,sig){

  a = dim(x)
  n = a[1]
  p = a[2]
  siginv = solve(sig)
  C= matrix(0,p,p);  #Sum(x-u)(x-u)t
  sxm = matrix(0,p,1) #sum(x-u)
  for (i in 1:n){
    xm = x[i,] - mu
    sxm = sxm + xm
    C = C + xm %*% t(xm) }
    B = siginv %*% (-n*sig + 2*C)%*%siginv      #Used for DDSS
    V= t(sxm)%*%siginv                          #Used for DDMS/DDSM

##############################################DDmu########################################
  ddmm =-n*siginv
##############################################DDsigma#####################################
  ddss = matrix(0,p*(p+1)/2,p*(p+1)/2)
  r = 1
  for(i in 1:p){
    for(j in i:p){
      c = 1
      for(k in 1:p){
        for(l in k:p){
          if(i == j & k == l ){
            ddss[r, c] = (-1/2)*B[k,i]%*%siginv[i,k]}
          if(i == j & k != l){
            ddss[r, c] = (-1/2)*(B[k,j]%*%siginv[i,l] + B[l,i]%*%siginv[j,k]) }
          if(i != j & k == l){
            ddss[r, c] = (-1/2)*(B[k,j]%*%siginv[i,l] + B[k,i]%*%siginv[j,l])}
          if(i != j & k != l){
            ddss[r, c] =  (-1/2)*(B[k,j]%*%siginv[i,l] + B[k,i]%*%siginv[j,l]
                          + B[l,j]%*%siginv[i,k] + B[l,i]%*%siginv[j,k]) }
          c = c +1
          if( c == ((p*(p+1))/2)+1){
            r = r+1}
             } } } }
  #######################################Diagonals##########################################
  ddms = matrix(0,p,p*(p+1)/2)

  for(r in 1:p){
    c = 1;
    for(k in 1:p){
      for(l in k:p){
        if(k == l){
          ddms[r,c]= -siginv[r,k]*V[k];
          }
        else {
```

```
          ddms[r,c]=-(siginv[r,k]*V[l]+siginv[r,l]*V[k]);
        }
        c = c + 1;
      }}}

  ddsm = t(ddms)    #TRANSPOSE for other diagonal

  ######################### BLOCK MATRIX #####################################

  A = cbind(ddmm,ddms)
  B = cbind(ddsm,ddss)

  hess=rbind(A,B)

  ############################# OUTPUT #######################################
  list( ddmu = ddmm, ddsigma = ddss, ddmusigma = ddms, ddsigmamu=ddsm)

  return(hess)
}
```

FISHER INFORMATION MATRIX FUNCTION:

```
Fisher=function(x,mu,sig){

############################## SETUP ##################################################
  a = dim(x)
  n = a[1]
  p = a[2]
  siginv = solve(sig)
  C= matrix(0,p,p);  #Sum(x-u)(x-u)t
  sxm = matrix(0,p,1) #sum(x-u)
  for (i in 1:n){
    xm = x[i,] - mu
    sxm = sxm + xm
    C = C + xm %*% t(xm)
  }
#################################### MU #####################################################
  fm = n*siginv
###################################### DIAGONALS ############################################
  fms = matrix(c(0),p,p*(p+1)/2)
  fsm =t(fms)
######################################## Fisher of Sigma ####################################
  fss = matrix(c(0),p*(p+1)/2,p*(p+1)/2)    #Same matrix format as DDSS

  r = 1
  for(i in 1:p){
    for(j in i:p){
      c = 1
      for(k in 1:p){
        for(l in k:p){
          if(i == j & k == l ){
            fss[r, c] = (n/2)*(siginv[i,k]^2)
          }
```

```
        if(i == j & k != l){
          fss[r, c] = n*(siginv[k,j]*siginv[i,l]) }

        if(i != j & k == l){
          fss[r, c] = n*(siginv[k,i]*siginv[j,l])}

        if(i != j & k != l){
          fss[r, c] = n*(siginv[k,j]*siginv[i,l]+siginv[k,i]*siginv[j,l])}

        c = c +1
        if( c == ((p*(p+1))/2)+1){
          r = r+1}
           }}}
             }

  ############################## Block Matrix  #############################

  A = cbind(fm,fms)
  B = cbind(fsm,fss)

  fish=rbind(A,B)

  return(fish)
}
```

STEEPEST ASCENT ALGORITHM

```
optmvn <- function (x, mu, sig, maxit,tolerr,tolgrad) {

  cat("Iteration", " Halving", "  log-likelihood",    "||Gradient||\n")

  for (it in 1:maxit){

    halve=0
    a =likemvn(x,mu,sig,gcomp = TRUE)
    dsig = a$grads
    dm = a$gradm
    sig1 = sig + dsig
    mu1 = mu + dm

    sigvec=vech(sig)                         #vectorizing sigma for tolerr and tolgrad

    sig1vec=vech(sig1)                       #vectorizing sigma1 for tolerr and tolgrad

    theta=matrix(c(mu,sigvec),ncol=1)       #Vector of initial values
    thetab=matrix(c(mu1,sig1vec),ncol=1)    #Vector of updated values
    elnorm=norm(matrix(c(dm,vech(dsig)),ncol=1))  #Norm of the gradient
    relerr= norm(thetab-theta)/max(1,norm(thetab))  #Relative Error

    cat(sprintf(' %2.0f                 %4.4f   %1.1e\n'
      ,it,a$l,elnorm))

    if(relerr < tolerr & elnorm < tolgrad){
      break
```

```r
    }

    while (min(eigen(sig1)$value)<=0)   #Half stepping for positive  covariance matrix
    {
      halve = halve + 1
      sig1 = sig + dsig/2^halve
      cat(sprintf(' %2.0f        %2.0f       %s\n'
                  ,it,halve,'   NA'))
    }

   mu1 = mu + dm/2^halve                        #Getting the same step for mu
   atmp = likemvn(x,mu1,sig1,gcomp = FALSE)

    while (atmp$l < a$l){      #Halfstepping for direction
      halve = halve+1
      mu1 = mu + dm/2^halve
      sig1 = sig + dsig/2^halve
      atmp = likemvn(x,mu1,sig1,gcomp = TRUE)
      elnorm=norm(matrix(c(atmp$gradm,vech(atmp$grads)),ncol=1))
      cat(sprintf(' %2.0f        %2.0f       %4.4f  %1.1e\n'
                  ,it,halve,atmp$l,elnorm)) }

                 mu = mu1                  #Update Values
                 sig= sig1

    cat(sprintf('%s\n'
              ,'_____________________________________________'))
  }

  list(mean = mu, covariance= sig)
}
```

```r
optmvn(datan, mu_0,sigma_0,1000,tolerr,tolgrad)
```

OUTPUT:

```
Iteration  Halving   log-likelihood ||Gradient||
  1                -1402.1983   2.1e+03
  1         1          NA
  1         2          NA
  1         3          NA
  1         4          NA
  1         5          NA
  1         6          NA
  1         7          NA
  1         8          NA
  1         9          NA
_____________________________________________
  2                -934.4694   4.8e+02
  2         1          NA
  2         2          NA
  2         3          NA
  2         4          NA
  2         5          NA
```

```
  2         6          NA
  2         7          NA
  2         8       -870.9406  6.2e+02
-------------------------------------------
  .         .           .         .
  .         .           .         .
  .         .           .         .

 998                -707.1835   2.9e-05
 998         1      -707.1835  2.7e-02
 998         2      -707.1835  1.4e-02
 998         3      -707.1835  6.9e-03
 998         4      -707.1835  3.4e-03
 998         5      -707.1835  1.7e-03
 998         6      -707.1835  8.3e-04
 998         7      -707.1835  4.0e-04
 998         8      -707.1835  1.9e-04
 998         9      -707.1835  7.8e-05
 998        10      -707.1835  2.5e-05
-------------------------------------------
 999                -707.1835   2.5e-05
 999         1      -707.1835  2.3e-02
 999         2      -707.1835  1.2e-02
 999         3      -707.1835  5.8e-03
 999         4      -707.1835  2.9e-03
 999         5      -707.1835  1.4e-03
 999         6      -707.1835  7.0e-04
 999         7      -707.1835  3.4e-04
 999         8      -707.1835  1.6e-04
 999         9      -707.1835  6.6e-05
 999        10      -707.1835  2.1e-05
-------------------------------------------
 1000                -707.1835   2.1e-05
 1000         1      -707.1835  1.9e-02
 1000         2      -707.1835  9.7e-03
 1000         3      -707.1835  4.8e-03
 1000         4      -707.1835  2.4e-03
 1000         5      -707.1835  1.2e-03
 1000         6      -707.1835  5.9e-04
 1000         7      -707.1835  2.8e-04
 1000         8      -707.1835  1.3e-04
 1000         9      -707.1835  5.5e-05
 1000        10      -707.1835  1.7e-05
-------------------------------------------
$mean
           [,1]
[1,]  0.8921026
[2,] -1.1205652
[3,]  1.8870434

$covariance
          [,1]      [,2]      [,3]
[1,] 0.9397384 0.6475741 0.5840537
[2,] 0.6475741 1.1196468 0.6617128
```

```
[3,] 0.5840537 0.6617128 0.8365201
```

The Steepest Ascent algorithm goes through the maximum amount of iterations without breaking the Gradient/MRE threshold, but we do ultimately converge to a desired MLE.

FISHER SCORING ALGORITHM:

```
optmvnFisher <- function (x, mu, sig, maxit, tolerr, tolgrad) {

  a = dim(x)
  n = a[1]
  p = a[2]
  p1 = p+1
  p2 = p+p*(p+1)/2

  cat("Iteration", " Halving", "  log-likelihood",    "||Gradient||\n")

  for (it in 1:maxit){
    halve=0
    a =likemvn(x,mu,sig,gcomp = TRUE)

    sigma=vech(sig)                   #vectorized sigma
    theta=matrix(c(mu,sigma),ncol=1)  # vector consisting of all mu's and sigmas.

    dm = a$gradm
    dsig = a$grads
    dsigvec=vech(dsig)                #vectorized dsig for the norm

    f= Fisher(x,mu,sig)
    finv=solve(f)

    gradtheta=matrix(c(dm,dsigvec),ncol=1)     #vectorized gradient
    elnorm=norm(gradtheta)

    thetab = theta + finv%*%gradtheta    # Steepest Ascent

    relerr= norm(thetab-theta)/max(1,norm(thetab))  #Relative Error


    cat(sprintf(' %2.0f                %4.4f   %1.1e\n'
            ,it,a$l,elnorm))

    if(relerr<tolerr & elnorm<tolgrad){        #Checking Accuracy
      break }

    #Getting sigma back into a matrix to check for positive eigenvalues

    sig1 = thetab[p1:p2]    # Pulling all of the sigma values out of the vector

    sig1 = xpnd(sig1,p)     # creating a symmetric covariance matrix

    halve=0
    while (min(eigen(sig1)$value)<=0)
    {
```

```
      halve = halve + 1
      thetab = theta + (finv%*%gradtheta)/2^halve

      cat(sprintf(' %2.0f        %2.0f       %s\n'
                  ,it,halve,'   NA'))

      sig1 = thetab[p1:p2]                     #updated vector to matrix
      sig1 = xpnd(sig1,p)
    }

    mu1  = thetab[1:p]
    atmp = likemvn(x,mu1,sig1,gcomp =FALSE)

    while (atmp$l < a$l & halve <= 20){   #Checking direction
      halve = halve+1
      thetab = theta + (finv%*%gradtheta)/2^halve

      mu1=thetab[1:p]                     #updated vector to matrix
      sig1 = thetab[p1:p2]
      sig1=xpnd(sig1,p)

      atmp = likemvn(x,mu1,sig1,gcomp =TRUE)

      dm = atmp$gradm
      dsig = atmp$grads
      dsigvec=vech(dsig)

      gradtheta=matrix(c(dm,dsigvec),ncol=1)
      elnorm2=norm(gradtheta)

      cat(sprintf(' %2.0f        %2.0f       %4.4f   %1.1e\n'
                  ,it,halve, atmp$l, elnorm2)) }

    if(halve==0){           #Printing iteration out if while loop doesn't start
       atmp = likemvn(x,mu1,sig1,gcomp =TRUE)
       dm = atmp$gradm
       dsig = atmp$grads
       dsigvec=vech(dsig)
       gradtheta=matrix(c(dm,dsigvec),ncol=1)
        elnorm2=norm(gradtheta)
    cat(sprintf(' %2.0f        %2.0f       %4.4f   %1.1e\n'
                ,it,halve, atmp$l, elnorm2))
    }

       mu = mu1
       sig = sig1

    cat(sprintf('%s\n'
                ,'_____________________________________________'))
  }

  list(mean = mu, covariance= sig)
}
```

```r
optmvnFisher(datan,mu_0,sigma_0,1000,tolerr,tolgrad)
```

```
## Iteration  Halving   log-likelihood ||Gradient||
##   1                 -1402.1983   2.1e+03
##   1         0       -896.5452  4.0e+01
## ---------------------------------------------
##   2                 -896.5452   4.0e+01
##   2         0       -707.1835  7.4e-12
## ---------------------------------------------
##   3                 -707.1835   7.4e-12

## $mean
## [1]  0.8921026 -1.1205652  1.8870434
##
## $covariance
##           [,1]      [,2]      [,3]
## [1,] 0.9397384 0.6475741 0.5840537
## [2,] 0.6475741 1.1196468 0.6617128
## [3,] 0.5840537 0.6617128 0.8365201
```

The Fisher algorithm converges to the same MLE as our Steepest Ascent, but at a much faster convergence rate. We note that it took a mere 3 iterations to arrive at our desired estimate.

NEWTON METHOD ALGORITHM

```r
optmvnNewton <- function (x, mu, sig,maxit,tolerr,tolgrad) {

  a = dim(x)
  n = a[1]
  p = a[2]
  p1 = p+1
  p2 = p+p*(p+1)/2

  cat("Iteration", " Halving", "  log-likelihood",    "||Gradient||\n")

  for (it in 1:maxit){
    halve=0
    a =likemvn(x,mu,sig,gcomp =TRUE)

    sigmavec=vech(sig)
    theta=matrix(c(mu,sigmavec),ncol=1)

    dm = a$gradm
    dsig = a$grads
    dsigvec=vech(dsig)

    h=hessian(x,mu,sig)
    hinv=solve(h)

    gradtheta=matrix(c(dm,dsigvec),ncol=1)

    elnorm=norm(gradtheta)

    thetab = theta - hinv%*%gradtheta   #Newton Method
    relerr= norm(thetab-theta)/max(1,norm(thetab))
```

```
    cat(sprintf(' %2.0f                 %4.4f   %1.1e\n'
                ,it,a$l,elnorm))

    if(relerr < tolerr & elnorm < tolgrad){
      break }

# Getting sigma into a matrix to check for positive eigenvalues

    sig1=thetab[p1:p2]
    sig1=xpnd(sig1,p)
    mu1=thetab[1:p]

    halve=0
    while (min(eigen(sig1)$value)<=0)   #Parameter Check
    {
      halve = halve + 1
      thetab = theta - (hinv%*%gradtheta)/2^halve

      cat(sprintf(' %2.0f        %2.0f       %s\n'
                  ,it,halve,'   NA'))

      sig1 = thetab[p1:p2]
      sig1 = xpnd(sig1,p)
      mu1=thetab[1:p] }

    atmp = likemvn(x,mu1,sig1,gcomp = FALSE)

    while (atmp$l < a$l & halve <= 20){

      halve = halve+1
      thetab = theta - (hinv%*%gradtheta)/2^halve
      mu1=thetab[1:p]

      sig1=thetab[p1:p2]
      sig1=xpnd(sig1,p)

      atmp = likemvn(x,mu1,sig1,gcomp = TRUE)
      dm = atmp$gradm
      dsig = atmp$grads
      dsigvec=vech(dsig)
      gradtheta=matrix(c(dm,dsigvec),ncol=1)
      elnorm=norm(gradtheta)
      cat(sprintf(' %2.0f        %2.0f       %4.4f   %1.1e\n'
                  ,it,halve,atmp$l,elnorm))}

    if(halve==0){
       atmp = likemvn(x,mu1,sig1,gcomp = TRUE)     #Output if while loop doesn't start.
       dm = atmp$gradm
       dsig = atmp$grads
       dsigvec=vech(dsig)
       gradtheta=matrix(c(dm,dsigvec),ncol=1)
       elnorm=norm(gradtheta)
    cat(sprintf(' %2.0f        %2.0f       %4.4f   %1.1e\n'
```

```
                 ,it,halve,atmp$l,elnorm))
    }
      mu = mu1
      sig = sig1
    cat(sprintf('%s\n'
                 ,'_____________________________________________'))
  }
  list(mean = mu, covariance= sig)
  }
```

```
optmvnNewton(datan,mu_n,sigma_n,1000,tolerr,tolgrad)
```

```
Iteration  Halving   log-likelihood ||Gradient||
  1                 -3258.5392   1.3e+04
  1         1          NA
  1         2          NA
  1         3          NA
  1         4          NA
  1         5          NA
  1         6          NA
_____________________________________________
  2                 -1335.8986   1.7e+04
  2         1          NA
  2         2          NA
_____________________________________________
  .         .            .         .
  .         .            .         .
  .         .            .         .
_____________________________________________
 11                 -707.1835   3.2e-03
 11         0       -707.1835  3.7e-08
_____________________________________________
 12                 -707.1835   3.7e-08
 12         0       -707.1835  1.1e-12
_____________________________________________
 13                 -707.1835   1.1e-12

$mean
[1]  0.8921026 -1.1205652  1.8870434

$covariance
          [,1]      [,2]      [,3]
[1,] 0.9397384 0.6475741 0.5840537
[2,] 0.6475741 1.1196468 0.6617128
[3,] 0.5840537 0.6617128 0.8365201
```

The Newton algorithm also converges to the desired MLE after a short amount of iterations. Although less effecient than the Fisher method, we still find the Newton method to be a more viable option than Steepest Ascent.

E.)

```
datan2 =gen(200, 3, c(-1,1,2), matrix(c(1,.9,.9,.9,1,.9,.9,.9,1),3))# New data
optmvn(datan2,mu_0,sigma_0,1000,tolerr,tolgrad) #Steepest Ascent with new data.
```

OUTPUT

```
Iteration  Halving   log-likelihood ||Gradient||
  1                -1387.6201       2.1e+03
  1          1        NA
  1          2        NA
  1          3        NA
  1          4        NA
  1          5        NA
  1          6        NA
  1          7        NA
  1          8        NA
  1          9        NA
---------------------------------------------
  2                -863.5420        6.9e+02
  2          1        NA
  2          2        NA
  2          3        NA
  2          4        NA
  2          5        NA
  2          6        NA
  2          7        NA
  2          8        NA
  2          9        NA
  2         10        NA
---------------------------------------------
  .           .          .          .
  .           .          .          .
  .           .          .          .
 998                 -502.8761   7.1e-02
 998          1      -505.8211  9.1e+02
 998          2      -503.5059  3.6e+02
 998          3      -503.0226  1.6e+02
 998          4      -502.9115  7.5e+01
 998          5      -502.8848  3.7e+01
 998          6      -502.8783  1.8e+01
 998          7      -502.8766  8.9e+00
 998          8      -502.8762  4.4e+00
 998          9      -502.8761  2.2e+00
 998         10      -502.8761  1.0e+00
 998         11      -502.8761  4.9e-01
 998         12      -502.8761  2.1e-01
 998         13      -502.8761  6.9e-02
 998         14      -502.8761  1.0e-02
---------------------------------------------
 999                 -502.8761   1.0e-02
 999          1      -502.8810  2.5e+01
 999          2      -502.8773  1.2e+01
 999          3      -502.8764  6.2e+00
 999          4      -502.8762  3.1e+00
```

```
 999         5       -502.8761  1.6e+00
 999         6       -502.8761  7.8e-01
 999         7       -502.8761  3.9e-01
 999         8       -502.8761  1.9e-01
 999         9       -502.8761  9.5e-02
 999        10       -502.8761  4.6e-02
------------------------------------------
 1000                 -502.8761   4.6e-02
 1000         1       -504.2234  5.1e+02
 1000         2       -503.1814  2.2e+02
 1000         3       -502.9490  1.0e+02
 1000         4       -502.8939  5.0e+01
 1000         5       -502.8805  2.4e+01
 1000         6       -502.8772  1.2e+01
 1000         7       -502.8764  6.0e+00
 1000         8       -502.8762  3.0e+00
 1000         9       -502.8761  1.5e+00
 1000        10       -502.8761  7.1e-01
 1000        11       -502.8761  3.3e-01
 1000        12       -502.8761  1.4e-01
 1000        13       -502.8761  4.8e-02
------------------------------------------
$mean
          [,1]
[1,] -1.1195134
[2,]  0.8731728
[3,]  1.8775656

$covariance
          [,1]      [,2]      [,3]
[1,] 0.9212101 0.8487487 0.7940290
[2,] 0.8487487 1.0310343 0.8448413
[3,] 0.7940290 0.8448413 0.8695619
```

This Steepest Ascent optimization still took the entire 1000 iterations to converge, but we can note that the final gradient value is larger than the final gradient value for our first Steepest Ascent optimization using the 0.7 covariance matrix. Thus,we can say that the first optimization (.7) converges to our value at a quicker rate than the second optimization (.9). We will use the eigenvalues of the information matrix to calculate the convergence rate and compare.

```r
#Create parameter values for MLE estimates

seven.mu=matrix(c( 0.8921026, -1.1205652, 1.8870434))
seven.sig=matrix(c(0.9397384, 0.6475741, 0.5840537,0.6475741,
               1.1196468, 0.6617128, 0.5840537, 0.6617128, 0.8365201),3,3)

nine.mu   =  matrix(c( -1.1195134, 0.8731728, 1.8775656))
nine.sig  = matrix(c(0.9212101, 0.8487487,0.7940290, 0.8487487, 1.0310343, 0.8448413,
                    0.7940290, 0.8448413,0.8695619 ),3)

#Compute Hessians
seven.hess = hessian(datan,seven.mu,seven.sig)
nine.hess  = hessian(datan2,nine.mu,nine.sig)

#Compute eigenvalues
```

```
ev7=eigen(-seven.hess)$val
ev9=eigen(-nine.hess)$val

#Convergence ratio
cr7 = ((max(ev7) - min(ev7))/(max(ev7) + min(ev7)))^2
cr9 = ((max(ev9) - min(ev9))/(max(ev9) + min(ev9)))^2
```

```
cr7
```

```
## [1] 0.9390445
```

```
cr9
```

```
## [1] 0.994833
```

Thus, we can observe that the convergence ratio from our first Steepest Ascent (.939) is smaller than that of the second Steepest Ascent Algorithm (.994). From the discussion in lecture, we can confirm that the First steepest ascent converges faster than the second steepest ascent.

F.) Standard errors can be found by using a modified version of our Fisher function.

```
Fisher2=function(x,mu,sig){
  a = dim(x)
  n = a[1]
  p = a[2]
  siginv = solve(sig)
  C= matrix(0,p,p);  #Sum(x-u)(x-u)t
  sxm = matrix(0,p,1) #sum(x-u)
  for (i in 1:n){
    xm = x[i,] - mu
    sxm = sxm + xm
    C = C + xm %*% t(xm)}

  fm = n*siginv
  fss = matrix(c(0),p*(p+1)/2,p*(p+1)/2)   #Same matrix format as DDSS
  r = 1
  for(i in 1:p){
    for(j in i:p){
      c = 1
      for(k in 1:p){
        for(l in k:p){
          if(i == j & k == l ){
            fss[r, c] = (n/2)*(siginv[i,k]^2)
          }
          if(i == j & k != l){
            fss[r, c] = n*(siginv[k,j]*siginv[i,l]) }
          if(i != j & k == l){
            fss[r, c] = n*(siginv[k,i]*siginv[j,l])}
          if(i != j & k != l){
            fss[r, c] = n*(siginv[k,j]*siginv[i,l]+siginv[k,i]*siginv[j,l])}
          c= c +1
          if( c== ((p*(p+1))/2)+1){
            r = r+1}
           }}}
          }
  list(fishermu =fm, fishersigma = fss)
}
```

We will use the MLE estimate found in part D to find their respective standard errors.

```
finalmu=matrix(c( 0.8921026, -1.1205652, 1.8870434))
finalsig=matrix(c(0.9397384, 0.6475741, 0.5840537,0.6475741,
                1.1196468, 0.6617128, 0.5840537, 0.6617128, 0.8365201),3,3)

final = Fisher2(datan,finalmu,finalsig)    #Fisher of estimates
se_mu = sqrt(diag(solve(final$fishermu)))  #Standard error or mu
se_sigma = sqrt(diag(solve(final$fishersigma))) #Standard error of sigma
se_sigma = xpnd(se_sigma,3)
```

```
se_mu
```

```
## [1] 0.06854701 0.07482135 0.06467303
```

```
se_sigma
```

```
##            [,1]       [,2]       [,3]
## [1,] 0.09397384 0.08577667 0.07507426
## [2,] 0.08577667 0.11196468 0.08289966
## [3,] 0.07507426 0.08289966 0.08365201
```

Thus, we have found our standard error values for $\hat{\mu}$, (0.06854701 0.07482135 0.06467303) and $\hat{\Sigma}$, (0.09397384, 0.08577667, 0.07507426, 0.11196468, 0.08289966, 0.08365201).

2

A.)

$$l(\theta) = \sum_{i=1}^{n} \big(w_i log(\lambda(t_i) exp(x_i^t \beta) exp(-\mu_i))\big) + \sum_{i=1}^{n} (1-w_i)(log(exp(-\mu_i)))$$

$$= \sum_{i=1}^{n} w_i log(\lambda(t_i)) + w_i(x^T \beta) - \mu$$

$$= \sum_{i=1}^{n} w_i \Big[log(\frac{\lambda(t_i)}{\Lambda(t_i)}) + log(\Lambda(t_i)) + x^T \beta \Big] - \mu$$

$$= \sum_{i=1}^{n} w_i \Big[log(\frac{\lambda(t_i)}{\Lambda(t_i)}) + log(\Lambda(t_i) exp(x^T \beta)) \Big] - \mu$$

$$= \sum_{i=1}^{n} w_i log(\frac{\lambda(t_i)}{\Lambda(t_i)}) + w_i log(\mu) - \mu$$

$$= \sum_{i=1}^{n} w_i log(\mu) - \mu + \sum_{i=1}^{n} w_i log(\frac{\lambda(t_i)}{\Lambda(t_i)})$$

B.)

Gradient Elements:

$$\frac{\partial l}{\partial \alpha} = \sum_{i=1}^{n} w_i log(t_i) - t_i^{\alpha} log(t_i) exp(\beta_0 + \delta_i \beta_1) + \frac{w_i}{\alpha} = \sum_{i=1}^{n} w_i log(t_i) - \mu_i log(t_i) + \frac{w_i}{\alpha}$$

$$\frac{\partial l}{\partial \beta_0} = \sum_{i=1}^{n} w_i - t_i^{\alpha} exp(\beta_0 + \delta_i \beta_1) = \sum_{i=1}^{n} w_i - \mu_i$$

$$\frac{\partial l}{\partial \beta_1} = \sum_{i=1}^{n} w_i \delta_i - t_i^{\alpha} \delta_i exp(\beta_0 + \delta_i \beta_1) = \sum_{i=1}^{n} \delta_i w_i - \delta_i \mu_i$$

Hessian elements:

$$\frac{\partial^2 l}{\partial \alpha^2} = \sum_{i=1}^{n} -t_i^{\alpha} log^2(t_i) exp(\beta_0 + \delta \beta_1) - \frac{w_i}{\alpha^2} = \sum_{i=1}^{n} -\mu_i log^2(t_i) - \frac{w_i}{\alpha^2}$$

$$\frac{\partial^2 l}{\partial \beta_0^2} = \sum_{i=1}^{n} -t_i^{\alpha} exp(\beta_0 + \delta_i \beta_1) = \sum_{i=1}^{n} -\mu_i$$

$$\frac{\partial^2 l}{\partial \beta_1^2} = \sum_{i=1}^{n} -t_i^{\alpha} \delta_i^2 exp(\beta_0 + \delta_i + \beta_1) = \sum_{i=1}^{n} -\delta^2 \mu_i$$

$$\frac{\partial^2 l}{\partial\alpha\partial\beta_0} = \sum_{i=1}^{n} -t_i^{\alpha} log(t_i) exp(\beta_0 + \delta_i\beta_1) = \sum_{i=1}^{n} -\mu_i log(t_i)$$

$$\frac{\partial^2 l}{\partial\alpha\partial\beta_1} = \sum_{i=1}^{n} -t_i^{\alpha}\delta_i log(t_i) exp(\beta_0 + \delta_i\beta_1) = \sum_{i=1}^{n} -\delta_i log(t_i)\mu_i$$

$$\frac{\partial^2 l}{\partial\beta_0\partial\beta_1} = \sum_{i=1}^{n} -t_i^{\alpha}\delta_i exp(\beta_0 + \delta_i\beta_1) = \sum_{i=1}^{n} -\delta_i\mu_i$$

We first need to input our data:

```
survival = c(6, 6, 6, 6, 7, 9, 10, 10, 11, 13, 16, 17, 19, 20, 22, 23, 25, 32, 32, 34, 35,
              1, 1, 2, 2, 3, 4, 4, 5, 5, 8, 8, 8, 8, 11, 11, 12, 12, 15, 17, 22, 23)

censored =  c(0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
              1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)

treatment =  c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
              0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

Likelihood/Gradient/Hessian:

```
likelihood<-function(theta,gcomp,hesscomp){
a = theta[1]
Bo=theta[2]
B1=theta[3]


d=treatment
w=censored
t=survival

mu = (t^a)*exp(Bo+d*B1) #defined in textbook.

l = sum(w*log(mu)-mu+w*log(a/t))

if(gcomp==TRUE) {
  gradient=matrix(0,3,1)

  gradient[1]=sum(w*log(t)-mu*log(t)+w/a) #da
  gradient[2]=sum(w-mu)                    #dBo
  gradient[3]=sum(d*w-d*mu)                #B1

}

 if(hesscomp==TRUE){
  hessian = matrix(0,3,3)
  hessian[1,1] = sum(-(mu*log(t)^2)-w/(a^2))    #Daa
  hessian[1,2] = hessian[2,1] = sum(-log(t)*mu)    #DaB0
  hessian[1,3] = hessian[3,1] = sum(-d*log(t)*mu) #DaB1
  hessian[2,2] = sum(-mu)                          #DB0B0
  hessian[2,3] = hessian[3,2] = sum(-d*mu)         #DB0B1
  hessian[3,3] = sum(-mu*d^2)                      #DB1B1
 }

list(l=l,grad=if(gcomp) gradient,hess=if(hesscomp)hessian)
}
```

NEWTON ALGORITHM:

```
# No need to input parameters seperately this time, just use a vectorized theta.

Newton <- function (theta,maxit,tolerr,tolgrad) {

  cat("Iteration", " Halving", "  log-likelihood",    "||Gradient||\n")
```

```r
for(it in 1:maxit){

a =likelihood(theta,TRUE,TRUE)
H=solve(a$hess)
theta1 = theta - H%*%a$grad
atmp = likelihood(theta,FALSE,FALSE)
elnorm = norm(a$grad)
relerr=norm(theta1-theta)/max(1,norm(theta1))

cat(sprintf(' %2.0f                 %4.4f   %1.1e\n'
            ,it,a$l,elnorm))

if (norm(a$grad)<tolgrad & relerr<tolerr ){
  break
}

halve = 0;
alpha=theta1[1]
while (atmp$l < a$l || theta1[1]<0){
  halve = halve+1
  theta1 = theta -H%*%a$grad/(2^halve)
  atmp = likelihood(theta1,TRUE,TRUE)
  elnorm = norm(atmp$grad)

  cat(sprintf(' %2.0f        %2.0f       %4.4f\n'
              ,it,halve,atmp$l,elnorm))
        }
theta = theta1

cat(sprintf('%s\n'
            ,'_____________________________________________'))
}

alpha_0 = theta[1]
beta_0 = theta[2]
beta_1 = theta[3]

list(alpha = alpha_0, beta0 = beta_0, beta1 = beta_1)
        }
```

We will use a starting theta of $\alpha = 1$, $\beta_0 = 2$, and $\beta_1 = 3$

```r
Newton(c(1,2,3),1000,tolerr=10e-6,tolgrad=10e-9)
```

```
## Iteration  Halving   log-likelihood ||Gradient||
##   1                 -54538.1323   2.7e+05
## _____________________________________________
##   2                 -20049.2240   1.0e+05
## _____________________________________________
##   3                 -7379.7531   3.7e+04
## _____________________________________________
##   4                 -2736.1327   1.3e+04
## _____________________________________________
##   5                 -1042.6178   4.9e+03
```

```
## _____________________________________________
##   6                  -430.1560   1.8e+03
## _____________________________________________
##   7                  -211.3002   6.4e+02
## _____________________________________________
##   8                  -135.7602   2.2e+02
## _____________________________________________
##   9                  -112.4796   7.3e+01
## _____________________________________________
##  10                  -107.1769   1.9e+01
## _____________________________________________
##  11                  -106.5922   2.5e+00
## _____________________________________________
##  12                  -106.5795   6.3e-02
## _____________________________________________
##  13                  -106.5795   4.3e-05
## _____________________________________________
##  14                  -106.5795   2.0e-11
```

```
## $alpha
## [1] 1.365758
##
## $beta0
## [1] -3.070704
##
## $beta1
## [1] -1.730872
```

The newton algorithm took 14 iterations to reach the final MLE value of -106.5795 with desired accuracy. We can also see the estimates of α , β_0 and β_1 to be 1.365758, -3.070704 and -1.730872

C.)

In order to use the built in function, we need to create a likelihood and gradient function seperately.

```r
likelihood2 = function(theta){
  a = theta[1]
  Bo=theta[2]
  B1=theta[3]

  d=treatment
  w=censored
  t=survival
  mu = (t^a)*exp(Bo+d*B1)
  l = sum(w*log(mu)-mu+w*log(a/t))
  return(l)
}

gradient2 = function(theta)
  {
a = theta[1]
Bo=theta[2]
B1=theta[3]

d=treatment
w=censored
```

```
t=survival

mu = (t^a)*exp(Bo+d*B1)
da=sum(w*log(t)-mu*log(t)+w/a)
dBo=sum(w-mu)
dB1=sum(d*w-d*mu)
gradient=cbind(da,dBo,dB1)
return(gradient)
}
```

Using the BFGS Method, we have the following input: Note: control needs to be set to -1 in order to search for the maximum likelihood rather than the minimum. We will use the thta value (1,2,3) as an initial starting point.

```
optim(par=c(1,2,3),fn=likelihood2,gr=gradient2,method="BFGS",control=c(fnscale=-1))
```

```
## $par
## [1]  1.365748 -3.070683 -1.730874
##
## $value
## [1] -106.5795
##
## $counts
## function gradient
##       67       20
##
## $convergence
## [1] 0
##
## $message
## NULL
```

The estimates provided by the built in function are strinkingly similar to those that were computed using the Newton Algorithm.

D.) We will use the Hessian to find our standard errors, as the function has already been created and is the information matrix.

```
theta_mle=c(1.365748,-3.070683,-1.730874)
H=likelihood(theta_mle,FALSE,TRUE)$hess
invH=solve(-H)
se=sqrt(diag(invH))
```

```
se
```

```
## [1] 0.2011640 0.5580675 0.4130834
```

Thus, the above values are the standard errors for the MLE estimates of α, β_0 and β_1.

In order to compare the correlation of our estimates, we will use the following code:

```
cor(invH)
```

```
##             [,1]       [,2]        [,3]
## [1,]  1.00000000 -0.9830777 -0.07119053
## [2,] -0.98307775  1.0000000 -0.11273819
## [3,] -0.07119053 -0.1127382  1.00000000
```

We can see that α is highly negatively correlated with β_0 with a correlation of -0.983, and only slightly negatively correlated with β_1, having a correlation of -0.07. In addition, the correlation between β_0 and β_1 -0.112 which is quite low as well.